SISTEMA DE GESTION DE MAQUETAS Y ALGORITMO DE MOVIMIENTO EN LABERINTOS PARA LA ESCUELA DE SISTEMAS

202100543 – HAROLD BENJAMIN OXLAJ MANGANDI

Resumen

El Ministerio de Educación de Guatemala ha invitado a todas las universidades del país a participar en un concurso para desarrollar soluciones mecatrónicas innovadoras.
El concurso consiste en crear un dispositivo que sea capaz de recolectar objetivos que estarán ubicados en una maqueta rectangular especial donde existirán caminos y paredes. Los caminos estarán de forma horizontal o vertical y el dispositivo debe ser colocado en la entrada predefinida de la maqueta.

Por parte de la Escuela de Sistemas se desea crear un algoritmo capaz de identificar un camino lineal sin reutilizar segmentos de caminos anteriores para recolectar "N" objetivos en un orden predeterminado.

El proyecto presentado se basa en la solución a este problema utilizando como base programación orientada a objetos y TDAs lineales, se desarrolló una interfaz en consola para el manejo de las maquetas cargadas, así como también graficar el camino a recorrer por medio de graphviz.

Palabras clave
movimiento, automatización, gestión, objetos, laberinto.

Abstract

The Ministry of Education of Guatemala has invited all universities in the country to participate in a competition to develop innovative mechatronic solutions. The competition involves creating a device capable of collecting objectives located on a special rectangular model, which will have paths and walls. The paths will be arranged horizontally or vertically, and the device must be placed at the predefined entrance of the model.

The School of Systems aims to create an algorithm capable of identifying a linear path without reusing segments from previous paths to collect "N" objectives in a predetermined order.

The project presented is based on solving this problem using object-oriented programming and linear ADTs as a foundation. A console interface was developed for managing the loaded models and for illustrating the path to be followed using Graphviz.

Keywords
movement, automation, management, objects, maze.

Introducción

La mecatrónica es una rama multidisciplinaria de la Ingeniería que desarrolla dispositivos y tecnologías. Esta área combina varios campos del conocimiento, integrando sistemas, electrónica, mecánica y control.

El principal objetivo de este proyecto no es la mecatrónica per se, sino el algoritmo que se empleará dentro de este campo para lograr las funcionalidades deseadas.

La base del proyecto fue la utilización de la Programación Orientada a Objetos y TDA's lineales, como lo son las listas enlazadas y las colas. El sistema cuenta con una interfaz de usuario que opera a través de la consola que puede cargar las maquetas que quiera por medio de un archivo XML, graficar las maquetas y simular el movimiento del dispositivo para alcanzar los objetivos.

Desarrollo del tema

El Primer vistazo que tendrá el usuario final con el sistema creado será una interfaz bastante clara y concisa, con un menú variado de opciones.

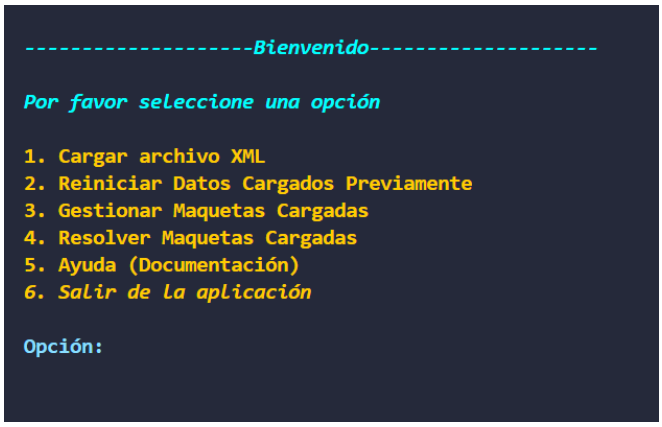


Figura 1. Menú Principal

Fuente: elaboración propia 2024

Al seleccionar la primera opción se le pedirá al usuario el nombre del archivo XML en el escritorio para cargar las maquetas, las maquetas no se sobrescriben al cargar archivos distintos, tienen permanencia.

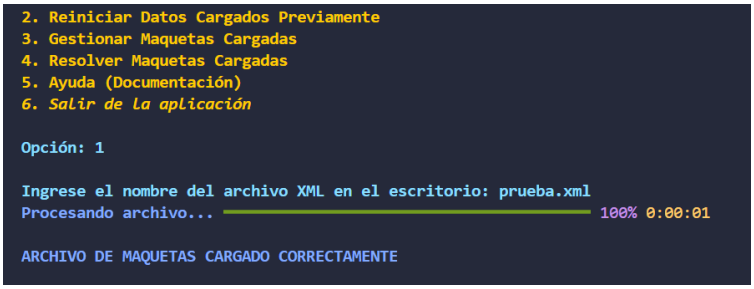
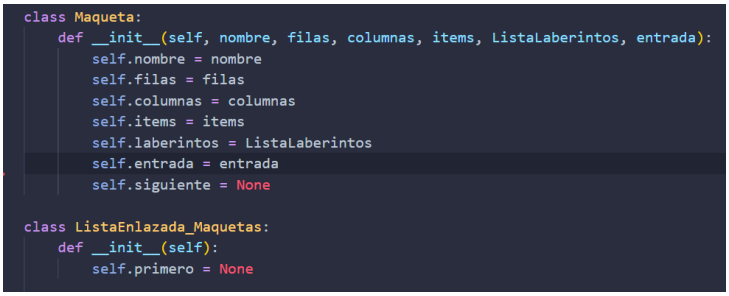


Figura 2. Carga de Archivo

Fuente: elaboración propia 2024

Las maquetas se guardan en una lista enlazada simple que contiene objetos Maqueta.



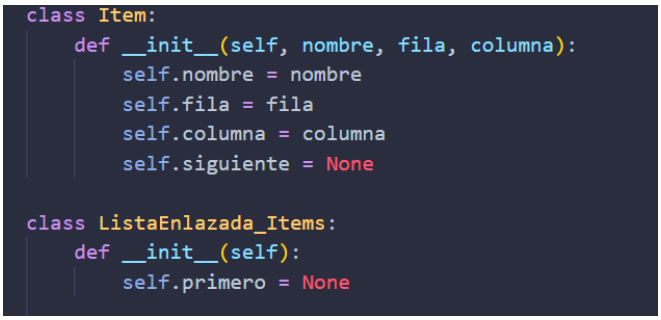
```
class Maqueta:
    def __init__(self, nombre, filas, columnas, items, ListaLaberintos, entrada):
        self.nombre = nombre
        self.filas = filas
        self.columnas = columnas
        self.items = items
        self.laberintos = ListaLaberintos
        self.entrada = entrada
        self.siguiente = None

class ListaEnlazada_Maquetas:
    def __init__(self):
        self.primero = None
```

Figura 3. Lista Enlazada de Maquetas

Fuente: elaboración propia 2024

Al igual que las maquetas los objetivos se guardan en una lista enlazada solo que esta simula una cola.



```
class Item:
    def __init__(self, nombre, fila, columna):
        self.nombre = nombre
        self.fila = fila
        self.columna = columna
        self.siguiente = None

class ListaEnlazada_Items:
    def __init__(self):
        self.primero = None
```

Figura 4. Lista Enlazada de items

Fuente: elaboración propia 2024

Al seleccionar la segunda opción se reiniciarán las maquetas cargadas, lo que dejará la aplicación como nueva.

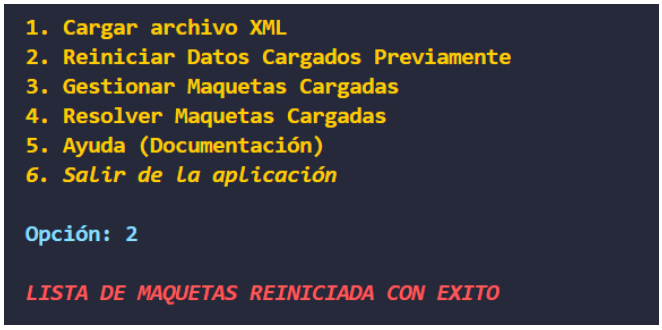


Figura 5. Reinicio de aplicación

Fuente: elaboración propia 2024

Al seleccionar la tercera opción se llevará a un submenú con tres opciones:

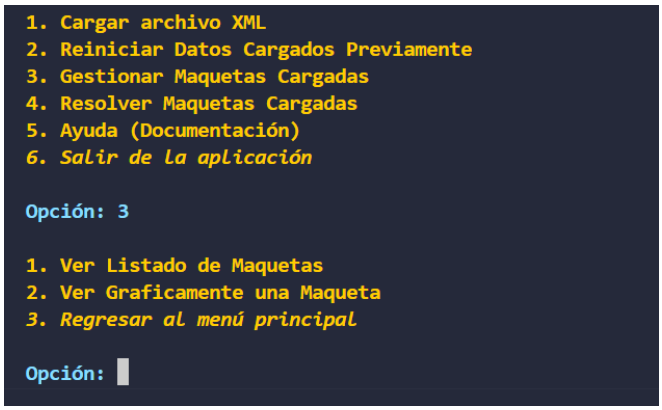


Figura 6. Submenú de Maquetas

Fuente: elaboración propia 2024

a. Al seleccionar la primera opción del submenú se imprimirá por medio de la consola los nombres de las maquetas cargadas hasta el momento en orden alfabético

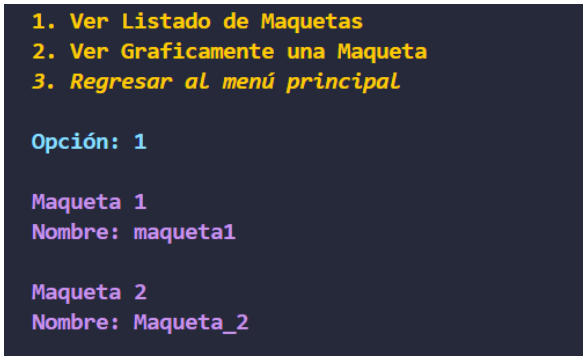


Figura 7. Submenú – Opción 1

Fuente: elaboración propia 2024

b. Al seleccionar la segunda opción se deberá de ingresar el nombre de la maqueta que se quiera ver gráficamente. Se generará un archivo .pdf.

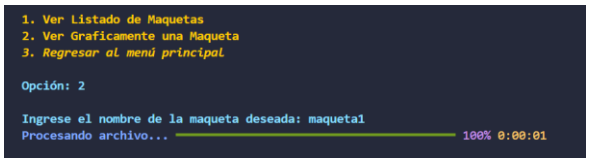


Figura 8. Submenú – Opción 2

Fuente: elaboración propia 2024

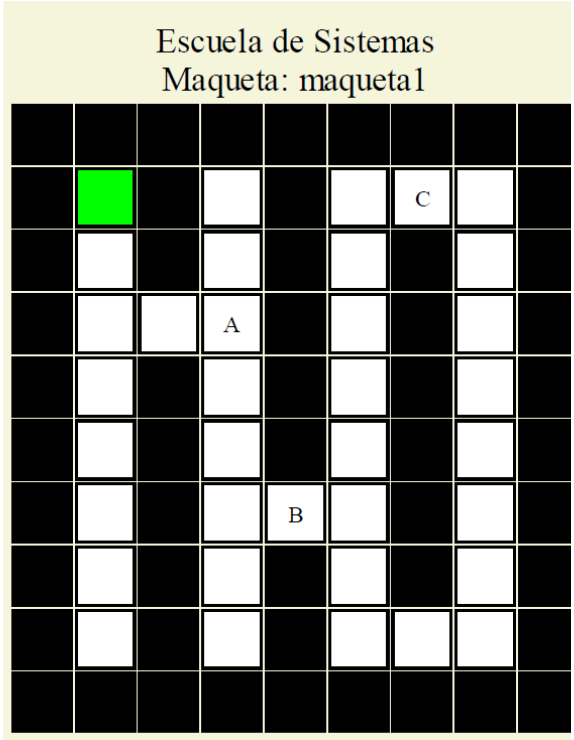


Figura 9. Grafica de una maqueta

Fuente: elaboración propia 2024

c. Al seleccionar la tercera opción simplemente regresará al Menú Principal.

Al seleccionar la cuarta opción se permitirá al usuario ingresar el nombre de la maqueta que se desee ver el camino recorrido y si es que tiene solución o no.

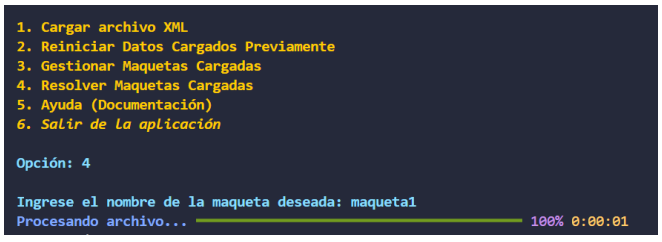


Figura 10. Ingresando nombre de la maqueta

Fuente: elaboración propia 2024

El camino seguido se mostrará de color celeste y la entrada de color verde.

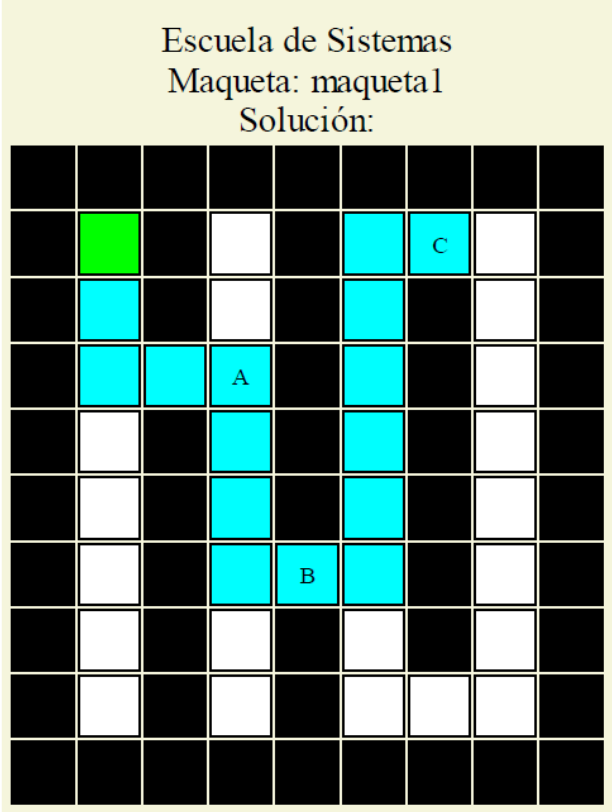


Figura 11. Solución grafica de la maqueta

Fuente: elaboración propia 2024

Los Laberintos son guardados en una lista enlazada y esta lista es la que contiene el algoritmo para el movimiento

```
class ListaEnlazada_Laberintos:
    def __init__(self):
        self.primero = None
        self.ultimo = None

    def esta_vacia(self):
        return self.primero is None
```

Figura 12. Lista Enlazada Laberintos

Fuente: elaboración propia 2024

El algoritmo de movimiento se basa en la posición inicial del nodo en la entrada, luego verifica la posición de los cuatro nodos adyacentes, arriba, abajo, izquierda y derecha por medio de su fila y columna luego, dependiendo de la posición del objetivo a buscar el nodo se moverá. Teniendo en cuenta que no puede moverse por nodos que ya haya visitado o que son paredes. Todo el algoritmo está contenido dentro de un bucle while que hasta que el ultimo objetivo sea alcanzado se seguirá movimiento, aunque si no encuentra nada o no se puede mover hay una condición que evitara que se vuelva un ciclo infinito que es un contador que disminuye mientras se repita el ciclo hasta llegar a 0 y en el momento que llegue a 0 parará el algoritmo. El algoritmo tiene en realidad dos funciones que en dado caso no encuentre todos los objetivos con las condiciones de una, pasará a la otra para intentar resolver el problema con sus nuevas condicionales dándole prioridad al movimiento a la izquierda

```
def movement(self, ListaOriginal, entrada, objetivos):
    if not self.esta_vacia():
        ListaOriginal_copia = self.copiar(ListaOriginal)
        entrada_copia = entrada
        objetivos_copia = objetivos.copiar(objetivos)
        global bloque_abajo
        global contador
        global bloque_arriba
        global bloque_derecha
        global bloque_izquierda
        global aux_laberinto
        global posicion_actual
        global aux2_laberinto
        contador = 100
        aux_laberinto = self.copiar(ListaOriginal)
        aux2_laberinto = self.copiar(ListaOriginal)
        objetivo_actual = objetivos.primero
        posicion_actual = aux_laberinto.get_coordenadas(entrada.fila, entrada.columna)
        while contador >0 :
            bloque_abajo = aux_laberinto.get_coordenadas(int(posicion_actual.fila)+1, int(posicion_act
            bloque_arriba = aux_laberinto.get_coordenadas(int(posicion_actual.fila)-1, int(posicion_ac
```

Figura 13. Algoritmo de movimiento - parte inicial de la función 1

Fuente: elaboración propia 2024

```
def movement_continue(self, ListaOriginal, entrada, objetivos):
    if not self.esta_vacia():
        global bloque_abajo
        global contador
        global bloque_arriba
        global bloque_derecha
        global bloque_izquierda
        global aux_laberinto
        global posicion_actual
        global aux2_laberinto
        contador = 100
        aux_laberinto = self.copiar(ListaOriginal)
        aux2_laberinto = self.copiar(ListaOriginal)
        objetivo_actual = objetivos.primero
        posicion_actual = aux_laberinto.get_coordenadas(entrada.fila, entrada.columna)
        while contador >0 :
            bloque_abajo = aux_laberinto.get_coordenadas(int(posicion_actual.fila)+1, int(posicion_act
            bloque_arriba = aux_laberinto.get_coordenadas(int(posicion_actual.fila)-1, int(posicion_ac
            bloque_derecha = aux_laberinto.get_coordenadas(int(posicion_actual.fila), int(posicion_act
            bloque_izquierda = aux_laberinto.get_coordenadas(int(posicion_actual.fila), int(posicion_a
            if int(objetivo_actual.fila) == int(posicion_actual.fila) and int(objetivo_actual.columna)
```

Figura 14. Algoritmo de movimiento - parte inicial de la función 2

Fuente: elaboración propia 2024

Al seleccionar la quinta opción se imprimirá por medio de la consola los datos del estudiante, así como el repositorio en el cual está alojado el proyecto

```
1. Cargar archivo XML
2. Reiniciar Datos Cargados Previamente
3. Gestionar Maquetas Cargadas
4. Resolver Maquetas Cargadas
5. Ayuda (Documentación)
6. Salir de la aplicación

Opción: 5

HAROLD BENJAMIN OXLAJ MANGANDI
202100543
https://github.com/Benjamin-Mangandi/IPC2_Proyecto2_202100543.git
```

Figura 15. Datos del Estudiante

Fuente: elaboración propia 2024

Conclusiones

Algunos algoritmos requieren de complejidad en la lógica de programación y a veces de muchas condicionales que el programador debe prever para su buen funcionamiento.

Las soluciones pueden ser muy variadas, al usarse no solo listas simples si no doblemente enlazadas u ortogonales, pero requiere un grado más de experiencia de parte del programador como capacidad para optimizar el rendimiento y la

eficiencia del código, así como en la anticipación y manejo de posibles errores.

Es importante añadir alguna verificación en dado caso la simulación no tuviera solución y el programa tuviese que tener alguna salida como en el caso de un bucle.

Anexos

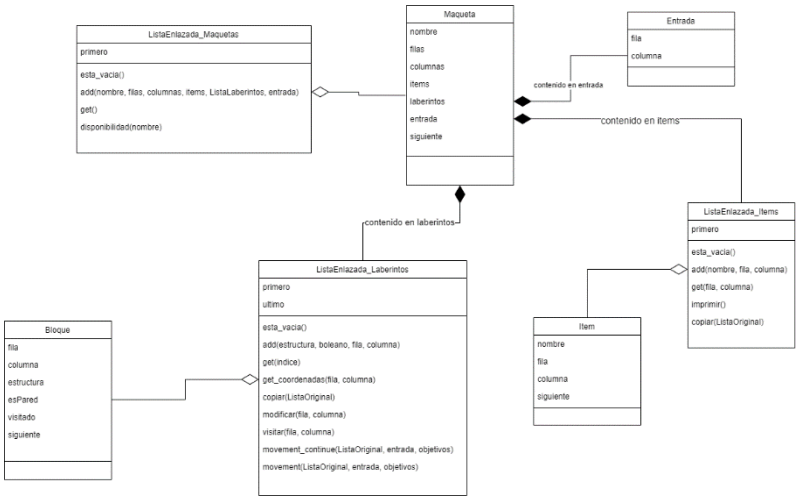


Figura 16. Diagrama de Clases de la solución
Fuente: elaboración propia 2024

Referencias bibliográficas

Label. Graphviz. (2022, Junio 12). Recuperado el[15/03/2024],de https://graphviz.org/docs/attrs/label/

Miller, B., & Ranum, D. (n.d.). Implementación de una cola en Python. Solución de problemas con algoritmos y estructuras de datos. Recuperado el [16/03/2024],de https://runestone.academy/ns/books/published/pythoned/BasicDS/ImplementacionDeUnaColaEnPython.html

Salcedo, L. (2018, 1 de julio). Linked List: Listas enlazadas - Implementación en Python. Mi Diario Python. Recuperado el [15/03/2024], de https://pythondiario.com/2018/07/linked-list-listas-enlazadas.html