



## **Pontificia Universidad Javeriana**

**Departamento de Ingeniería de Sistemas**

**Estructura de Datos**

**Segunda Entrega - Proyecto Estructura de Datos**

**Nombre de integrantes:**

**Tomás Figueroa Sierra**

**Santiago Camilo Rey Benavides**

**Juan Manuel Sánchez Bermúdez**

**Profesora: Andrea Rueda**

**Abril 25, 2023**

**Bogotá, Colombia**

## Contenido

<b>1. Corrección entrega 1:</b>	<b>3</b>
1.1. Documento diseño:	3
1.2. Plan de pruebas:	3
1.3. Código fuente:	3
<b>2. Diagrama de Relación:</b>	<b>5</b>
<b>3. TADs:</b>	<b>6</b>
3.1. TAD Análisis	6
3.2. TAD Movimiento	6
3.3. TAD Puntointeres	6
3.4. TAD Curiosity	7
3.5. TAD ArbolQuad	8
3.6. TAD NodoQuad	8
<b>4. Comandos:</b>	<b>10</b>
4.1. Comando: ayuda	10
4.2. Comando: cargar_comandos.	11
4.3. Comando: cargar_elementos.	12
4.4. Comando: agregar_movimiento.	13
4.5. Comando: agregar_analisis.	14
4.6. Comando: agregar_elemento.	15
4.7. Comando: guardar.	16
4.8. Comando: simular_comandos.	17
4.9. Comando: ubicar_elementos.	18
4.10. Comando: en_cuadrante	19
<b>5. Plan de pruebas:</b>	<b>20</b>
5.1. Plan de pruebas función simular comandos:	20
5.2. Plan de pruebas función en cuadrante:	23

# 1. Corrección entrega 1:

## 1.1. Documento diseño:

- Los contenedores deben indicar exactamente el tipo de dato que están almacenando.

Se cambiaron los tipos de datos en los contenedores del TAD Curiosity de modo que coincidieran exactamente con los tipos Análisis, Movimiento y Puntointeres.

- El diagrama de relación entre TADs no es un diagrama de clases, no debería existir diagrama de clases todavía porque en teoría aún no se ha implementado nada.

Ahora se incluye el diagrama de relación entre TADs.

- No se incluyen esquemáticos (diagramas de flujo, de actividad, de secuencia) para cada uno de los comandos del sistema.

Se incluyeron los diagramas de flujo para los comandos del sistema en este documento.

- No se incluye el análisis de entradas, salidas y condiciones para cada uno de los comandos del sistema.

El análisis de entradas salidas y condiciones para los comandos se incluye en este documento.

## 1.2. Plan de pruebas:

- Es importante complementar el plan de pruebas con pantallazos que evidencien la ejecución.

Se realizaron nuevos planes de pruebas con las especificaciones del enunciado 2 y se incluyeron los pantallazos de la ejecución.

## 1.3. Código fuente:

- Si los TADs no incluyen o están definidos con plantillas, no se deben implementar en archivos .hxx, sino en archivos .cxx o .cpp.

Los archivos de implementación se cambiaron a archivos cpp y cxx.

- En general, casi todos los comandos muestran información que no es necesaria en su ejecución. Lo único que deberían imprimir en pantalla los comandos son los mensajes indicados en el enunciado, de acuerdo a la situación particular.

Se eliminaron las salidas en pantalla con información innecesaria para los comandos de carga y de simulación, de modo que solo se muestra lo indicado. En el caso de los comandos de carga de comandos y elementos se muestra cuantos comandos y puntos fueron cargados del archivo de carga.

- No es necesario agregar 0 o 1 al inicio de cada línea del archivo de comandos para diferenciarlos entre movimiento y análisis. En teoría los tipos de movimiento y los tipos de análisis ya están bien definidos y no son tantos como para poder validarlos.

Se utilizaron las categorías de movimientos y análisis para clasificar los comandos y así evitar el uso de 0s y 1s en el archivo de comandos. A pesar de esto en el curiosity se sigue guardando un contenedor con el orden de los comandos para escribir el archivo generado por el comando guardar.

### 1.3.1. Comando Ayuda:

- La ayuda para cada comando no está disponible. Yo debería poder teclear ayuda guardar, y me debería salir en pantalla la información de cómo llamar al comando guardar y qué es lo que hace el comando.

Se modificó el comando ayuda y su función para que de información específica del comando tecleado como segundo argumento.

### 1.3.2. Comando cargar\_comandos:

- Si hay comandos que no son válidos, esa información debería mostrarse aquí al momento de cargar los comandos desde el archivo, y no al momento de la simulación.

Ahora la función del comando cargar\_comandos muestra en pantalla la línea del archivo que no contiene el formato de los comandos y posteriormente muestra cuantos comandos se agregaron de las líneas del archivo. Adicional a lo anterior, al momento de agregar el comandos a las listas de movimiento y análisis puede que los datos no correspondan al tipo de comando, si esto sucede se lee en pantalla “La información del análisis no corresponde a los datos esperados”, no se agrega ningún comando y se retorna false de la función agregar.

### 1.3.3. Comando cargar\_elementos:

- No parece estar funcionando el comando, genera violación de segmento cuando lo intenté probar con un archivo propio de prueba.

Es muy probable que haya pasado debido a los comentarios del archivo, pues no se estaban aceptando. Se cambió la función para agregar puntos de interés y de esta

manera acepta archivos con comentarios con la forma del ejemplo propuesto en el enunciado 2.

### 1.3.4. Comando agregar\_analisis:

- No me permite agregar un análisis sin comentario, cuando se supone que el comentario es opcional.

La corrección del comentario anterior soluciona el problema ocasionado por el comentario obligatorio, esto siempre y cuando este tecleado entre comillas simples.

## 2. Diagrama de Relación:

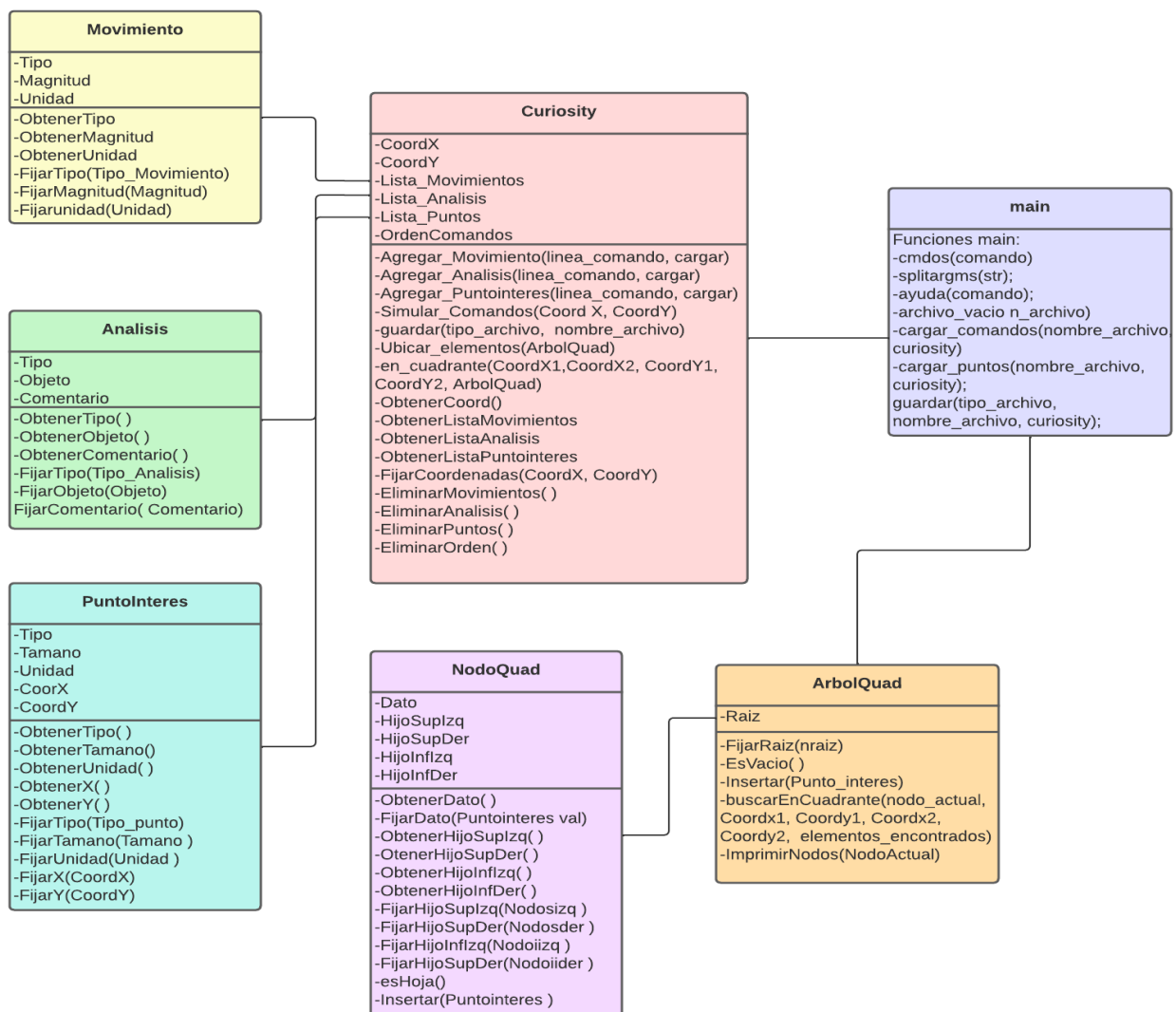


Figura 1. Diagrama de relación

## 3. TADs

### 3.1. TAD Análisis

#### Atributos:

- Tipo de análisis, cadena de caracteres, indica el tipo de análisis a realizar.
- Objeto, cadena de caracteres, se trata del objeto sobre el cual se hace el análisis.
- Comentario, cadena de caracteres, descripción e información adicional sobre el análisis.

#### Operaciones:

- Obtener\_tipo(), retorna el tipo de análisis.
- Obtener\_objeto(), retorna el objeto del análisis.
- Obtener\_comentario(), retorna el comentario del análisis.
- Fijar\_tipo (n\_tipo), fija un nuevo tipo de análisis como n\_tipo.
- Fijar\_objeto(n\_objeto), fija un nuevo objeto para el análisis como n\_objeto.
- Fijar\_objeto(n\_comentario), fija un nuevo comentario n\_comentario.

### 3.2. TAD Movimiento

#### Atributos:

- Tipo de movimiento, cadena de caracteres, indica si se trata de un giro o el avance del vehículo ("avanzar" o "girar").
- Magnitud, número real, es la magnitud (distancia o grados) del movimiento realizar.
- Unidad de medida, cadena de caracteres, establece la unidad de medida de la magnitud (cm o m).

#### Operaciones:

- Obtener\_tipo(), retorna el tipo de movimiento.
- Obtener\_magnitud (), retorna la magnitud del movimiento .
- Obtener\_unidad (), retorna la unidad de medida.
- Fijar\_tipo (n\_tipo), fija n\_tipo como nuevo tipo de movimiento.
- Fijar\_objeto(n\_objeto), fija n\_objeto como magnitud.
- Fijar\_unidad (n\_unidad), fija n\_unidad como unidad de medida.

### 3.3. TAD Puntointeres

#### Atributos:

- Tipo de elemento, carácter, carácter que indica de que se trata el punto de interés.

- Unidad de medida, cadena de caracteres, establece la unidad con la que se midió el objeto.
- Posición X, numero real, representa la magnitud de la coordenada del eje x.
- Posición Y, número real, representa la magnitud de la coordenada del eje y.

#### **Operaciones:**

- Obtener\_tipo(), retorna el tipo de terreno.
- Obtener\_unidad (), retorna la unidad de medida.
- Obtener\_coordenadas (), retorna la posición del punto de interés.
- Fijar\_tipo (n\_tipo), fija n\_tipo como tipo de terreno.
- Fijar\_unidad (n\_unidad), fija n\_unidad como unidad de medida.
- Fijar\_X (M), Establece M como coordenada en el eje X.
- Fijar\_Y (N), Establece N como coordenada en el eje N.

### **3.4. TAD Curiosity**

#### **Atributos:**

- Movimientos, lista de Movimiento, incluye los movimientos que debe realizar el vehículo manteniendo el orden en que fueron ingresados.
- Análisis, lista de Analisis, incluye los análisis con sus respectivos tipos que debe realizar el curiosity
- Puntos\_interes, lista de Puntosinteres, contiene los puntos de interés para el curiosity.
- Lista orden, lista de enteros, lista de 1s y 0s que indican el orden en que deben ser ejecutados los comandos de movimiento y análisis.
- Posición X, numero real, representa la magnitud de la coordenada del eje x.
- Posición Y, número real, representa la magnitud de la coordenada del eje y

#### **Operaciones:**

- Curiosity(), constructor del objeto curiosity, inicializa las coordenadas en el origen
- Simular\_coordenadas(), retorna el resultado de la posición del vehículo como simulación de los comandos de movimiento cargados y la posición indicada.
- Guardar(tipo\_archivo, nombre\_archivo), recibe el tipo de archivo y guarda los comandos del curiosity o los puntos de interés.
- ObtenerListaMovimientos(), retorna la lista de los movimiento cargados o agregados al curiosity;
- ObtenerListaAnalisis(), retorna la lista de tipo análisis con los objetos de este tipo cargados.
- ObtenerListaPuntos(), retorna la lista con los puntos de interés agregados o cargados al curiosity.
- ObtenerOrdenComandos(), retorna la lista de 1s y 0s con el orden de los comandos.
- Agregar\_Movimiento(cadena de caracteres), recibe una cadena de caracteres con los elementos del comando de movimiento, los tokeniza, crea un objeto de tipo Movimiento al que agrega estos elementos. Añade este elemento a la lista de movimientos y agrega un 0 a la lista de orden de comandos.
- Agregar\_Analisis(cadena de caracteres), recibe una cadena de caracteres con los elementos del comando de análisis, los tokeniza, crea un objeto de tipo

Análisis al que agrega estos elementos. Añade este elemento a la lista de movimientos y agrega un 1 a la lista de orden de comandos.

- Agregar\_Puntointeres(cadena de caracteres), recibe una cadena de caracteres con los elementos del puntos de interes, los tokeniza, crea un objeto de tipo Puntointeres al que agrega estos elementos. Añade este elemento a la lista de Puntointeres .
- fijarCoordenadas(numero real, numero real), fija las coordenadas actuales del curiosity.
- eliminarMovimientos(), vacía la lista de movimientos.
- eliminarAnalisis(); vacía la lista de Analisis.
- eliminarPuntos(), vacia la lista de Puntos.
- simular\_comandos(coordX, coordY), toma las coordenadas ingresadas y ejecuta los comandos cargados para mostrar en qué posición quedaría el vehículo.
- ubicar\_elementos(), ubica los elementos en la estructura de datos Quadtree
- en\_cuadrante(coordX1 coordX2 coordY1 coordY2), retorna una lista que esta dentro del cuadrante

### 3.5. TAD ArbolQuad

#### Atributos:

- raiz, tipo de dato nodo, crea la raíz del árbol.

#### Operaciones:

- ArbolQuad(), método constructor.
- ArbolQuad(val), crea un objeto de la clase ArbolQuad inicializando la data con el elemento pasado como parámetro.
- ArbolQuad(), método destructor.
- datoRaiz(), obtiene el dato que se encuentra en la raíz del árbol.
- obtenerRaiz(), retorna el dato que se encuentra en la raíz del árbol.
- fijarRaiz( n\_raiz),actualiza la raíz del árbol.
- esVacio(), evalúa si el árbol se encuentra vacío.
- buscarEnCuadrante(nodo\_actual, x1, y1, x2, y2, vector elementos\_encontrados), busca por todo el arbol si encuentra algun punto de interés dentro de los límites(x1, x2, y1, y2) y lo guarda en el vector elementos\_encontrados
- insertar(punto), inserta un nuevo punto de interes dentro del arbol, si esta vacia crea un nuevo y sino llama a la funcion insertar de NodoQuad
- imprimirNodos(nodo\_actual), imprime todos los valores de X y Y de cada uno de los nodos del arbol

### 3.6. TAD NodoQuad

#### Atributos:

- dato, dato de tipo entero, representa el dato que almacena el nodo.
- hijoSuplq, apuntador a NodoQuad, representa el hijo superior izquierdo del nodo.



- hijoSupDer, apuntador a NodoQuad, representa el hijo superior derecho del nodo.
- hijoInflzq, apuntador a NodoQuad, representa el hijo inferior izquierdo del nodo.
- hijoInfDer, apuntador a NodoQuad, representa el hijo inferior derecho del nodo.

#### **Operaciones:**

- NodoQuad(), método constructor.
- NodoQuad(val), crea un objeto de la clase NodoQuad inicializando la data con el elemento pasado como parámetro.
- NodoQuad(), método destructor.
- obtenerDato(), retorna el dato almacenado por el nodo.
- fijarDato(val), actualiza el dato almacenado por el nodo.
- obtenerHijoSuplZq(), retorna el hijo superior izquierdo del nodo.
- obtenerHijoSupDer(), retorna el hijo superior derecho del nodo.
- obtenerHijoInflZq(), retorna el hijo inferior izquierdo del nodo.
- obtenerHijoInfDer(), retorna el hijo inferior derecho del nodo.
- fijarHijoSuplZq(sizq), actualiza el hijo superior izquierdo del nodo con la data pasada como parámetro.
- fijarHijoSupDer(sder), actualiza el hijo superior derecho del nodo con la data pasada como parámetro.
- fijarHijoInflZq(iizq), actualiza el hijo inferior izquierdo del nodo con la data pasada como parámetro.
- fijarHijoInfDer(ider), actualiza el hijo inferior derecho del nodo con la data pasada como parámetro.
- esHoja(), identifica si el nodo es una hoja.
- insertar(punto), inserta un nuevo nodo dentro del arbol de acuerdo a los valores de X y Y del nuevo nodo

## 4. Comandos

### 4.1. Comando: ayuda

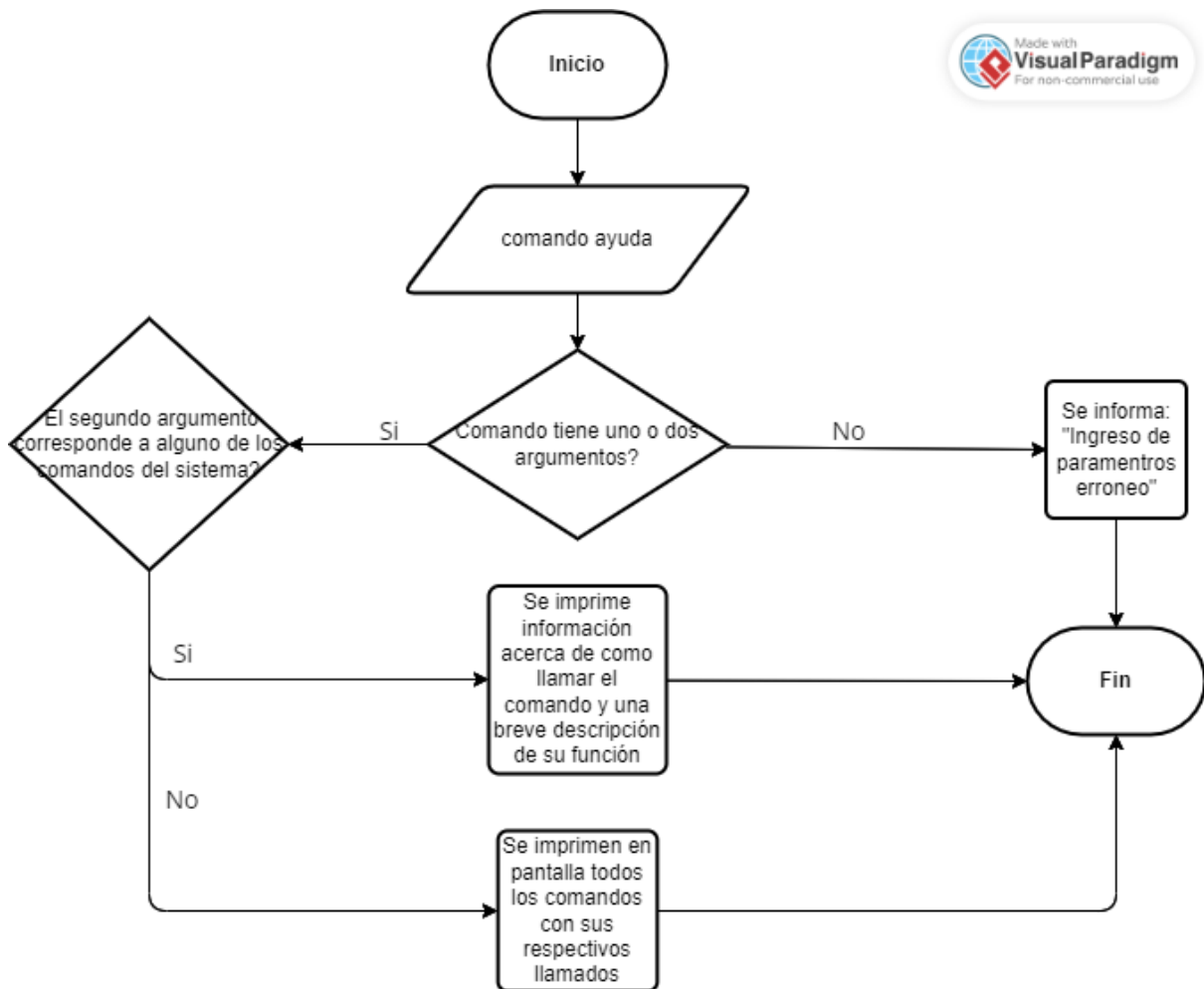


Figura 2. Diagrama de flujo comando ayuda

**Descripción:** muestra por consola una descripción del comando deseado.

**Entrada:** Comando “ayuda”.

**Salida:** Se tienen tres posibles salidas.

- Si los argumentos no se encuentran completos se le notifica al usuario.
- Si uno de los argumentos es erróneo se le notifica al usuario del error.
- Si la cantidad de argumentos y del comando se encuentran se le notifica el éxito mostrando el comando y una pequeña descripción.

## 4.2. Comando: cargar\_comandos.

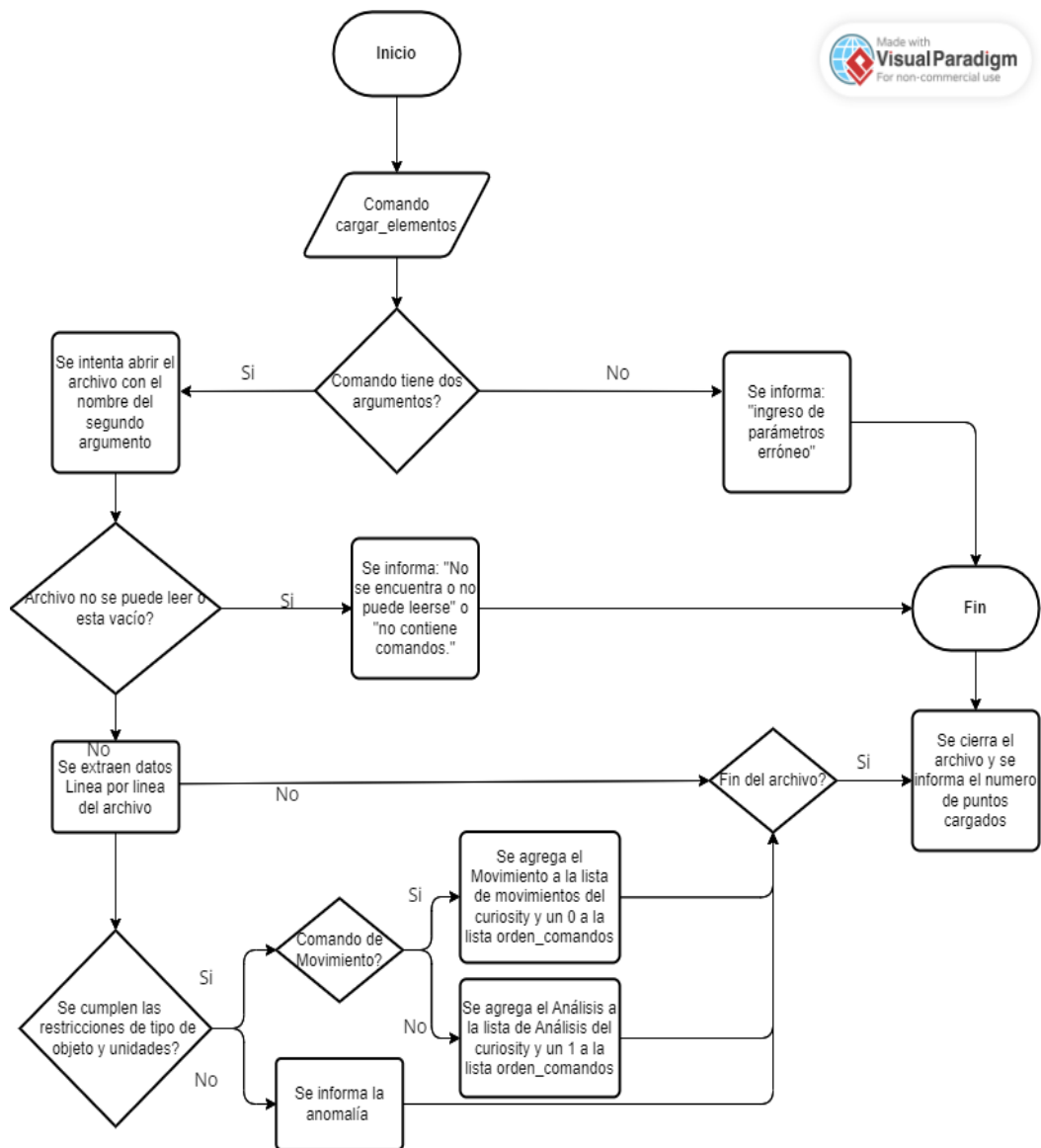


Figura 3. Diagrama de flujo comando cargar\_comandos

**Descripción:** Carga en memoria los comandos de análisis y movimiento del respectivo archivo.

**Entrada:** Se hace la lectura del respectivo archivo y se extrae los comandos tanto de análisis como de movimiento.

**Salida:** Se tienen dos posibles salidas.

- Si el archivo no existe o se encuentra vacío devuelve un mensaje de aviso al usuario.

- Si el archivo existe y cumple con los parámetros se carga los comandos de análisis y de movimiento avisando de su éxito al usuario.
- 

#### 4.3. Comando: cargar\_elementos.

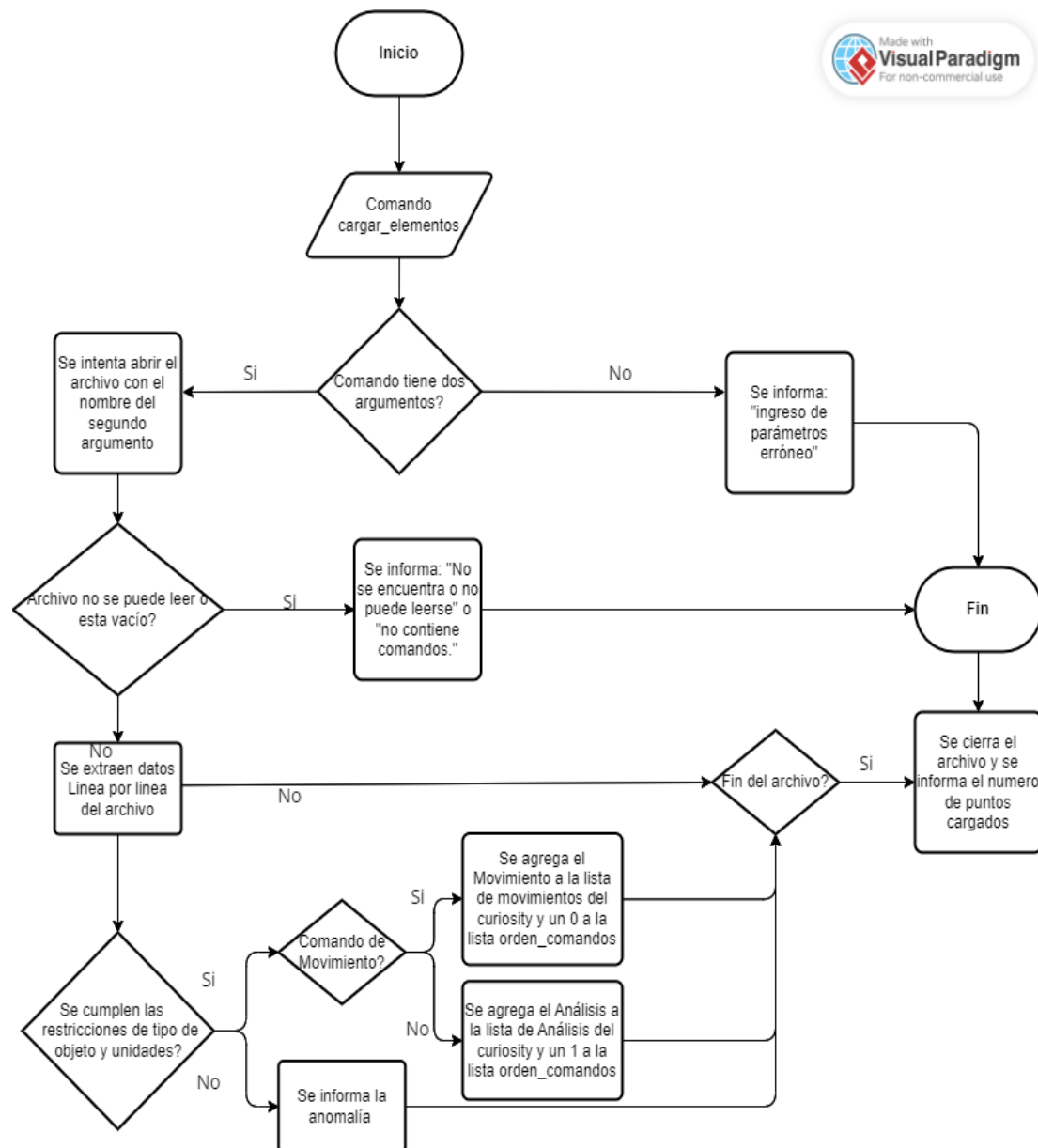


Figura 4. Diagrama de flujo comando cargar\_elementos

**Descripción:** Carga en memoria los comandos de cargar elementos que se encuentren en el respectivo archivo.

**Entrada:** Se hace la lectura del respectivo archivo y se extrae el o los elementos deseados existentes.

**Salida:** Se tienen dos posibles salidas

- Si el archivo no existe o se encuentra vacío devuelve un mensaje de aviso al usuario.
- Si el archivo existe y cumple con los parámetros se carga los elementos y posteriormente avisando de su éxito al usuario.

#### 4.4. Comando: agregar\_movimiento.

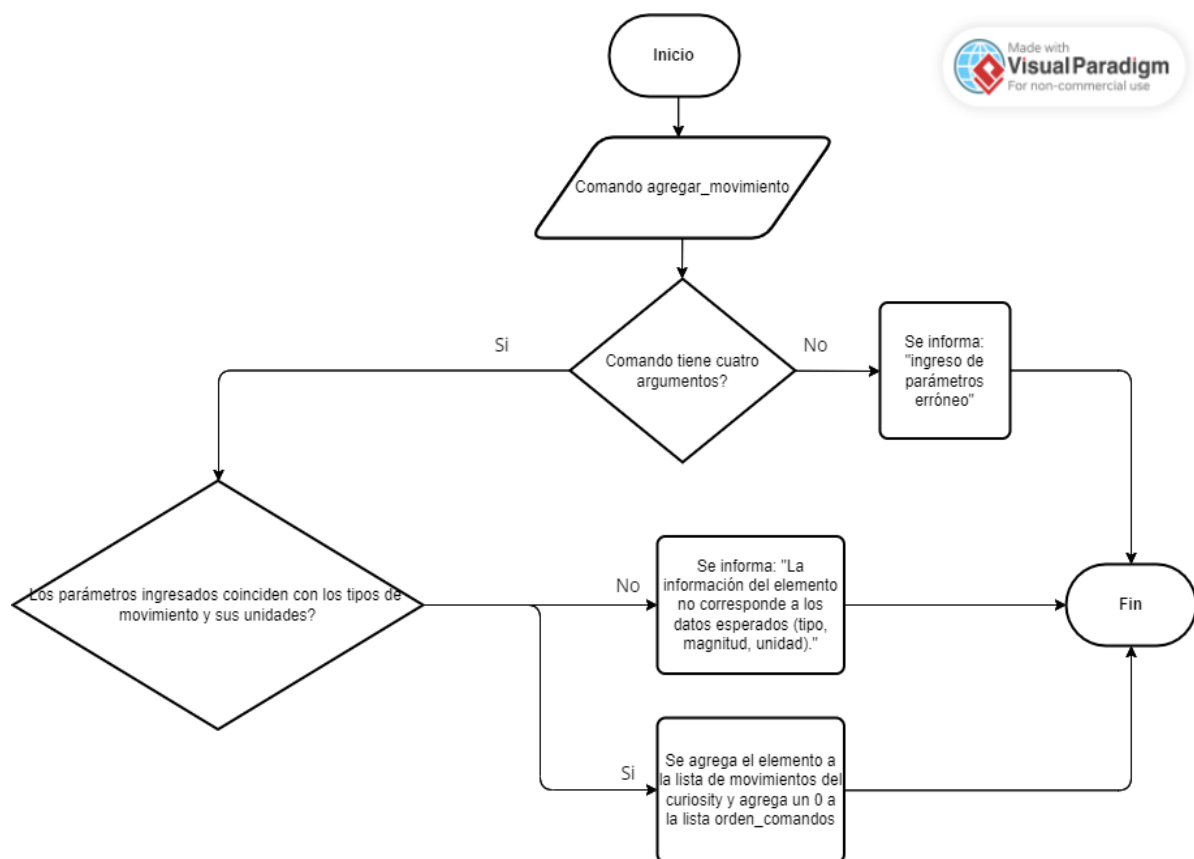


Figura 5. Diagrama de flujo comando agregar\_movimiento

**Descripción:** Carga en memoria el nuevo comando de movimiento ingresado por el usuario.

**Entrada:** Nuevo movimiento.

**Salida:** Se tienen dos posibles salidas.

- En el caso de que los datos estén incompletos o sean erróneos se le notificará al usuario.
- En el caso de que los datos sean correctos se procederá a agregarlos a la lista de movimientos notificando al usuario el éxito del proceso.

#### 4.5. Comando: agregar\_analisis.

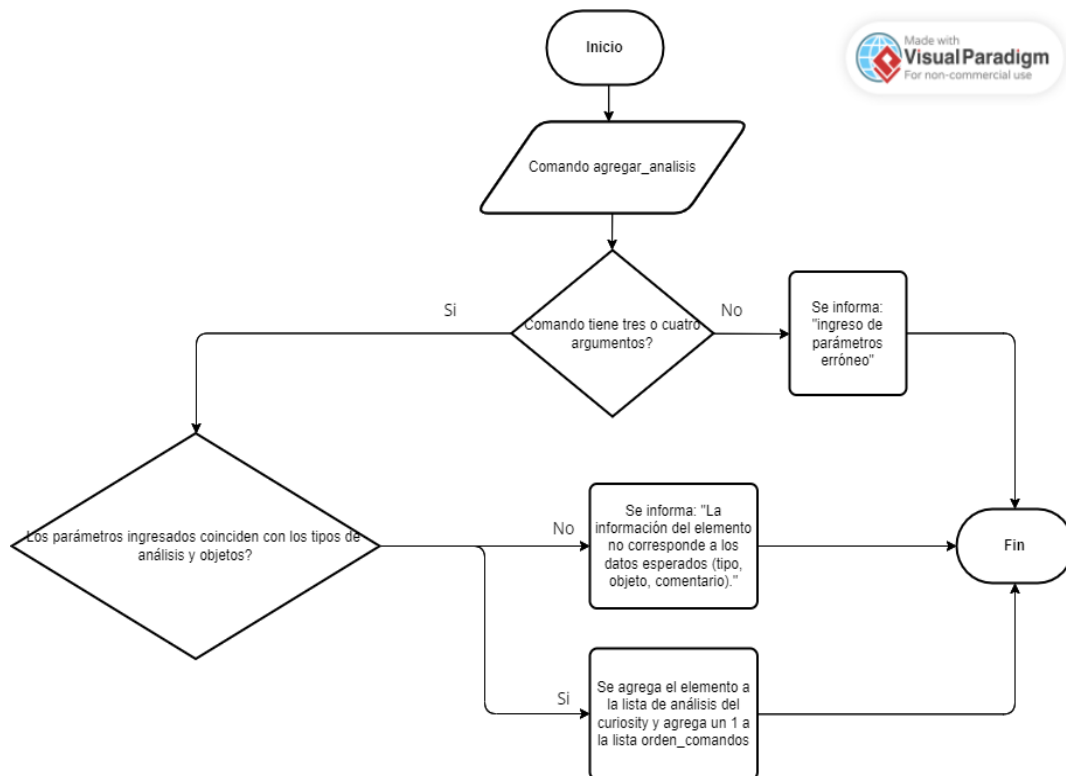


Figura 6. Diagrama de flujo comando agregar\_analisis

**Descripción:** Carga en memoria el nuevo comando de análisis ingresado por el usuario.

**Entrada:** Nuevo análisis.

**Salida:** Se tienen dos posibles salidas.

- En el caso de que los datos estén incompletos o sean erróneos se le notificará al usuario.
- En el caso de que los datos sean correctos se procederá a agregarlos a la lista de análisis notificando al usuario el éxito del proceso.

#### 4.6. Comando: agregar\_elemento.

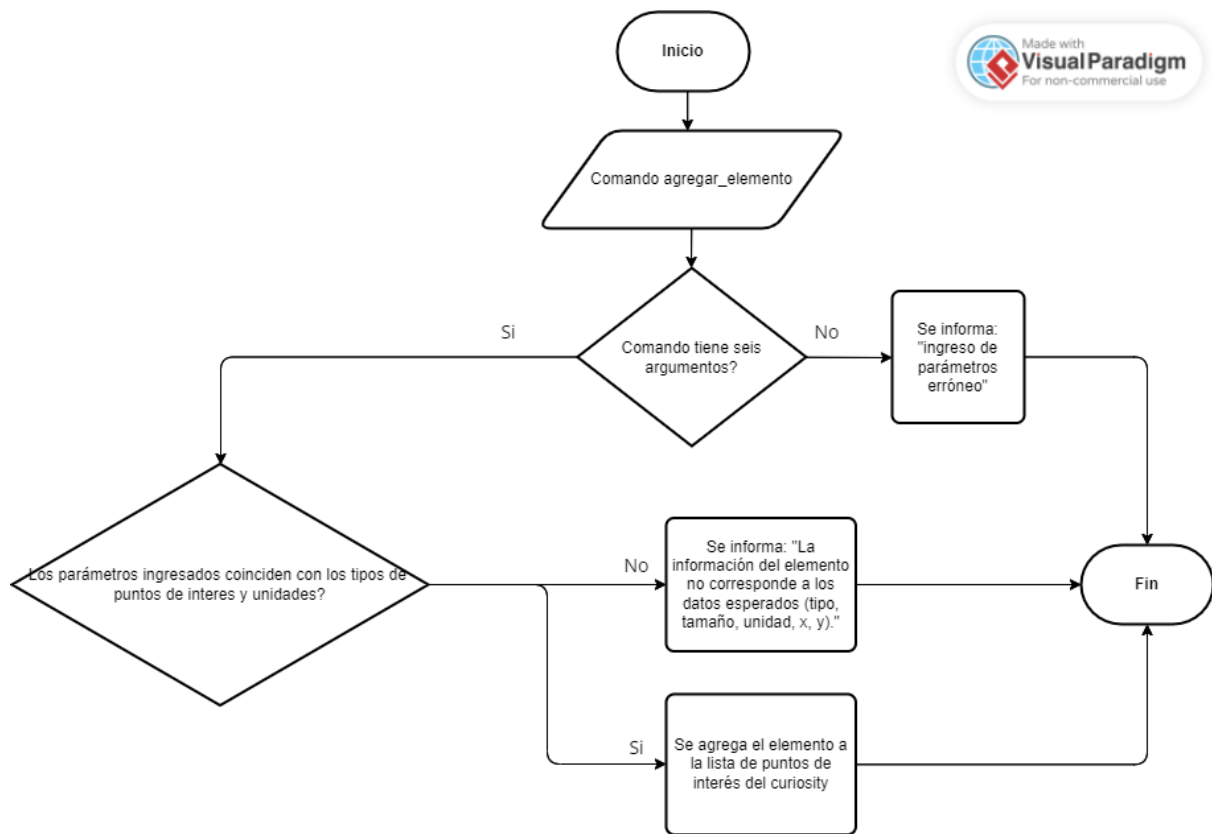


Figura 7. Diagrama de flujo comando agregar\_elemento

**Descripción:** Carga en memoria el nuevo elemento ingresado por el usuario.

**Entrada:** Nuevo elemento.

**Salida:** Se tienen dos posibles salidas.

- En el caso de que los datos estén incompletos o sean erróneos se le notificará al usuario.
- En el caso de que los datos sean correctos se procederá a agregarlos a la lista de puntos de interés del robot notificando al usuario el éxito del proceso.

#### 4.7. Comando: guardar.

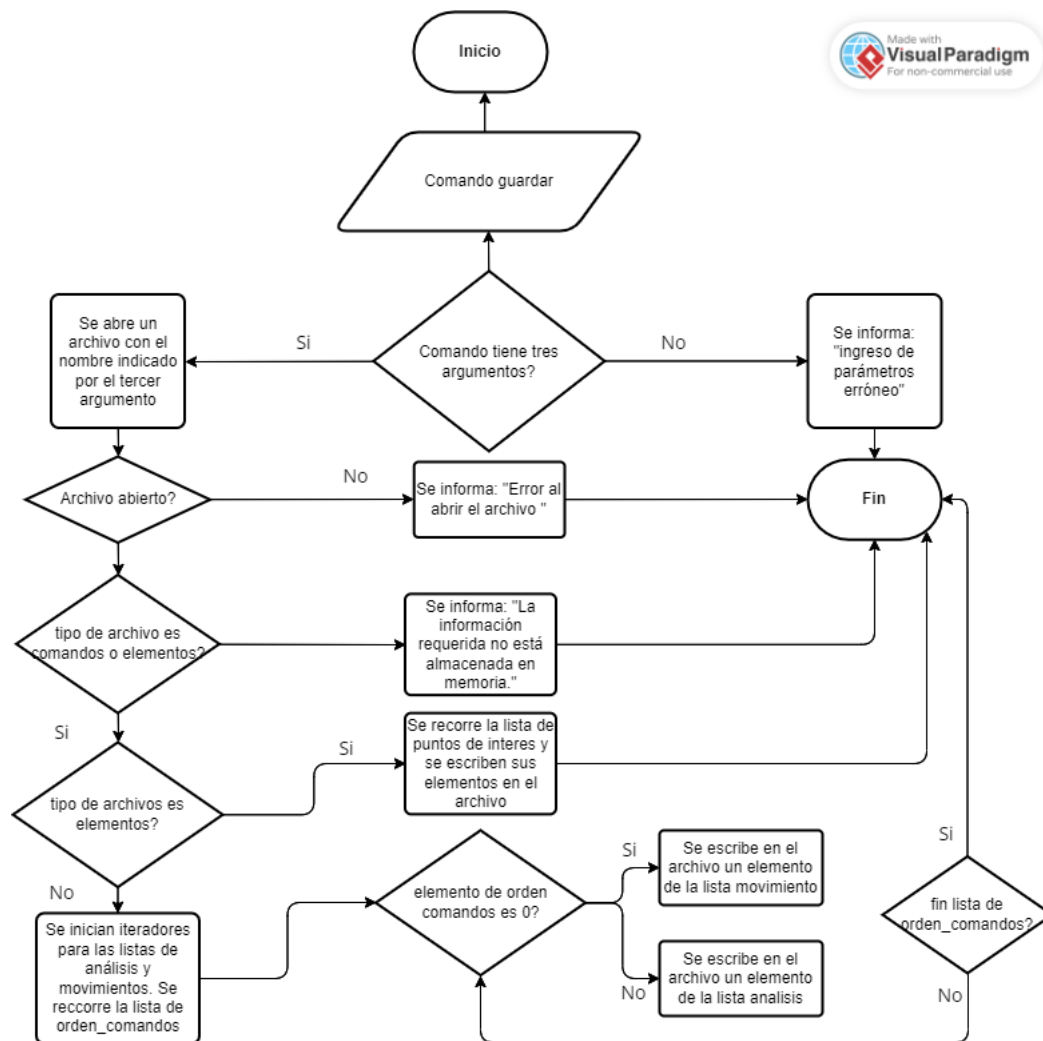


Figura 8. Diagrama de flujo comando guardar

**Descripción:** Guarda en un archivo la información correspondiente.

**Entrada:** Recibe el nombre que se le va a asignar y el tipo de archivo.

**Salida:** Se tienen tres posibles salidas.

- En el caso de que el archivo se encuentre abierto se le notifica al usuario del error.
- En el caso de que la extensión o que el nombre indicado sea incorrectos se le notifica al usuario del error.
- En el caso de que tanto el nombre como el tipo de archivo sean correcto y además se encuentre cerrado se le guardará la información en el archivo indicando el éxito al usuario.



#### 4.8. Comando: `simular_comandos`.

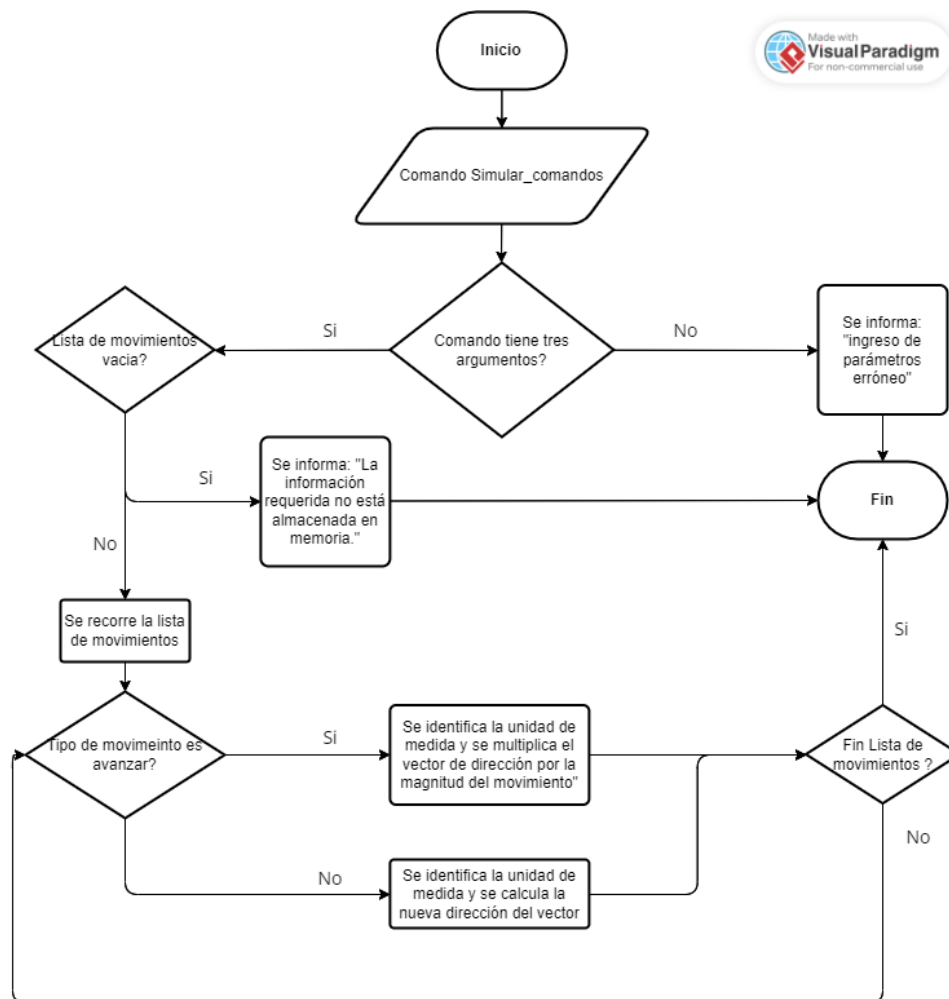


Figura 9. Diagrama de flujo comando `simular_comandos`

**Descripción:** Simula el comportamiento de los comandos de movimiento que se le enviarán al robot lo cual permite validar la nueva posición donde se encuentra ubicado.

**Entrada:** Coordenadas en “x” y en “y”.

**Salida:** Se tienen dos posibles salidas.

- En el caso de que los datos sean incorrectos o se encuentren incompletos se le notifica al usuario del error.
- En el caso de que los datos se encuentren completos sin errores se le notifica al usuario del éxito del proceso.

#### 4.9. Comando: ubicar\_elementos.

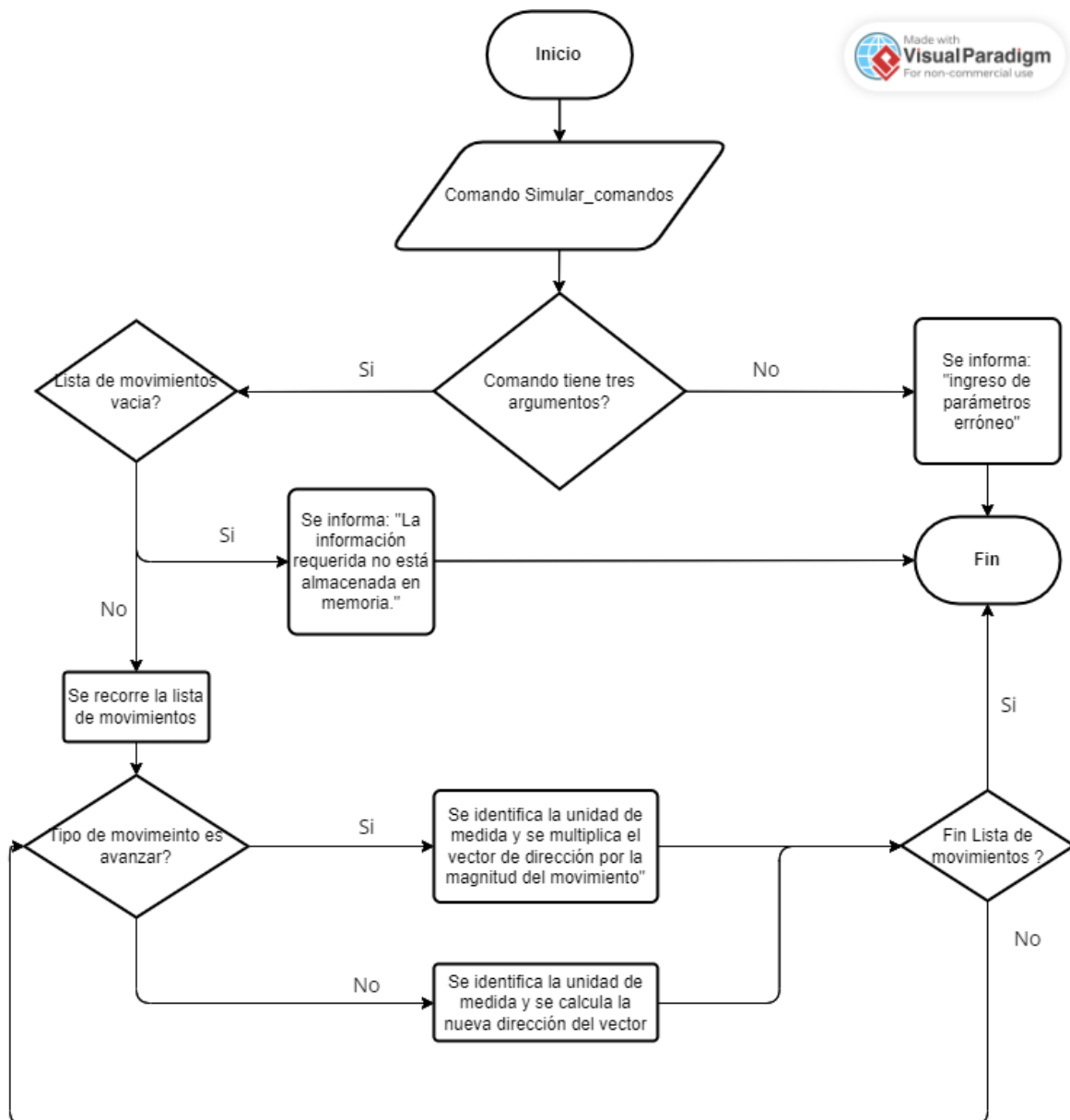


Figura 10. Diagrama de flujo comando ubicar\_elementos

**Descripción:** Utiliza la información de puntos de interés almacenados que permite luego realizar consultas geográficas sobre estos elementos.

**Entrada:** nombre comando.

**Salida:** Se tienen dos posibles salidas.

#### 4.10. Comando: en\_cuadrante

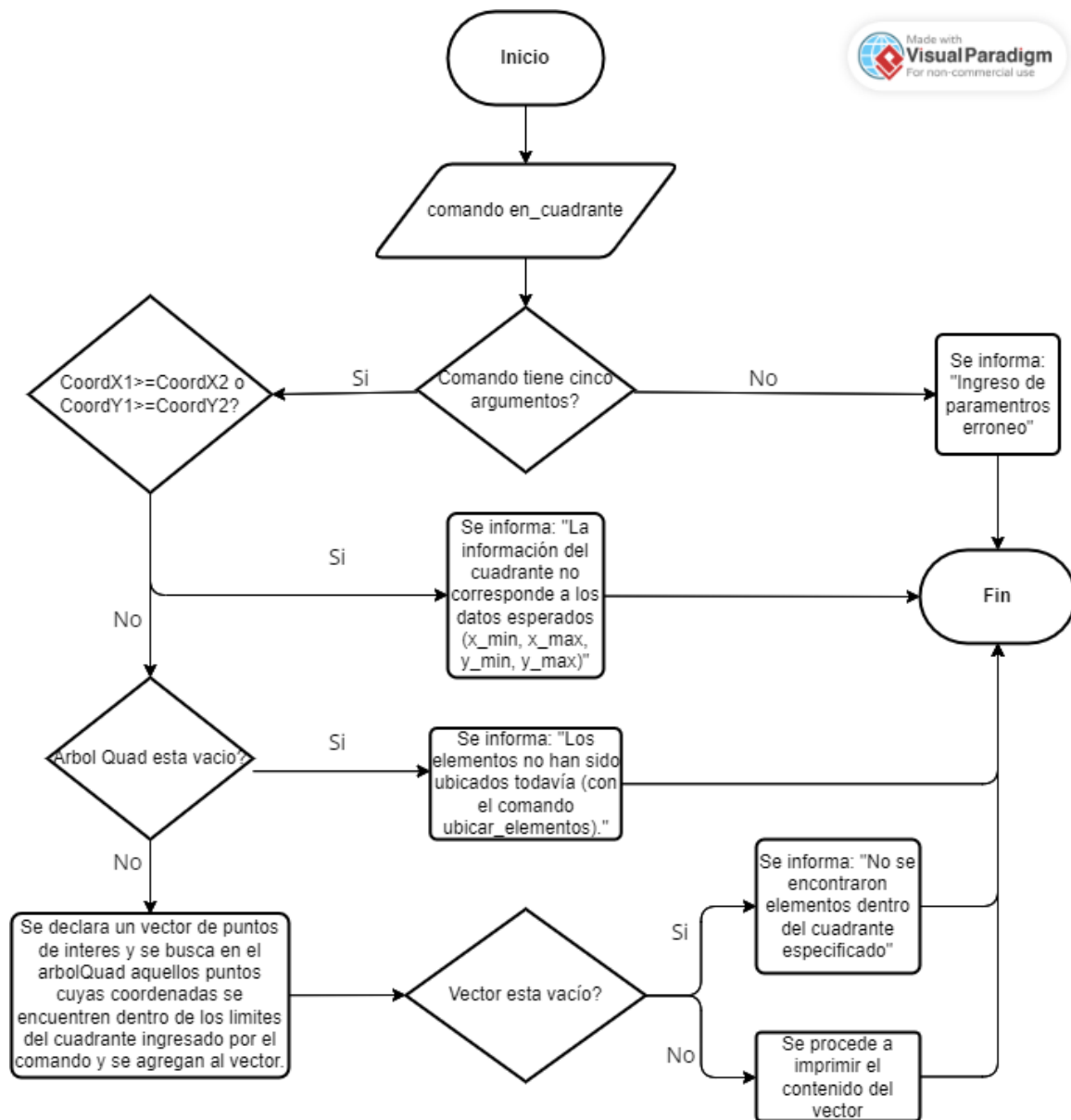


Figura 11. Diagrama de flujo comando en\_cuadrante

**Descripción:** Utiliza la estructura creada con el comando anterior para retornar una lista de los componentes o elementos que están dentro del cuadrante geográfico descrito por los límites de coordenadas en “x” y “y”.

**Entrada:** nombre comando y el conjunto de coordenadas, dos en “x” y dos “y”.

**Salida:** Se tienen dos posibles salidas.

- Retorna una lista de componentes utilizando la estructura del comando anterior.
- En el caso de que los datos sean erróneos se le notificará al usuario del error.

## 5. Plan de pruebas


### 5.1. Plan de pruebas función simular comandos:

Descripción plan de pruebas:

#### 5.1.1. Prueba 1: No hay información en el sistema

En esta prueba intentamos usar la función de simular comandos para comprobar que cuando no hubiera información de que movimientos realizar en el sistema, se imprimiera por pantalla un mensaje de error que comunicara al usuario de que no había información suficiente para simular comandos.

- **Valores ingresados por el usuario:** (0,0)
- **Salida esperada:** Mensaje de error (La información requerida no esta almacenada en memoria)
- **Evidencia de resultado:**



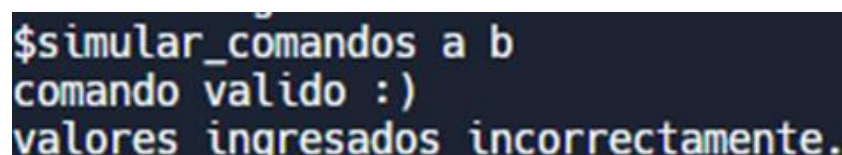
```
$simular_comandos 0 0
comando valido :)
La información requerida no está almacenada en memoria.
```

Figura 12. Evidencia prueba 1 simular\_comandos

#### 5.1.2. Prueba 2: Valores ingresados de manera incorrecta

En esta prueba intentamos usar la función de simular comandos para comprobar que pasaba si ingresábamos parámetros incorrectos, por ejemplo, si ingresábamos como parámetros letras en vez de números, esperando que por pantalla también nos muestre un mensaje de error

- **Valores ingresados por el usuario:** (a,b)
- **Salida esperada:** Mensaje de error (valores ingresados incorrectamente)
- **Evidencia de resultado:**



```
$simular_comandos a b
comando valido :)
valores ingresados incorrectamente.
```

Figura 13. Evidencia prueba 2 simular\_comandos

#### 5.1.3. Prueba 3: Valores ingresados correctamente

En esta prueba intentaremos usar el comando después de haber almacenado en memoria la información necesaria y utilizar valores correctos para llamarla.

- **Información cargada desde memoria:**

```

1  avanzar 40 cm
2  girar 45 grd
3  girar 45 grd
4  avanzar 10 m
5  girar -90 grd
6  avanzar 5 cm

```

Figura 14. Evidencia prueba 3 información cargada `simular_comandos`

- **Valores ingresados por el usuario:** (0,0)
- **Salida esperada:** (0.45, 10)
- **Evidencia de resultado:**

```

$simular_comandos 0 0
comando valido :)
La simulación de los comandos, a partir de la posición (0,0), deja al robot en la nueva posición (0.45,10).
Unidades en metros.

```

Figura 15. Evidencia prueba 3 `simular_comandos`

#### 5.1.4. Prueba 4: Valores ingresados correctamente desde posiciones negativas

En esta prueba intentaremos usar el comando ya habiendo almacenado en memoria la información necesaria y empezando desde posiciones negativas

- **Información cargada desde memoria:**

```

1  avanzar 4 m
2  girar 90 grd
3  avanzar 10 m
4  girar -180 grd
5  avanzar 20 m

```

Figura 16. Evidencia prueba 4 información cargada `simular_comandos`

- **Valores ingresados por el usuario:** (-5,-5)
- **Salida esperada:** (-1, -15)
- **Evidencia de resultado:**

```
$simular_comandos -5 -5
comando valido :)
La simulación de los comandos, a partir de la posición (-5,-5), deja al robot en la nueva posición (-1,-15).
Unidades en metros.
```

Figura 17. Evidencia prueba 4 simular\_comandos

### 5.1.5. Prueba 5: Valores ingresados correctamente desde posiciones positivas

En esta prueba intentaremos usar el comando, ya con la información cargada en memoria y empezando la simulación desde posiciones positivas

- **Información cargada desde memoria:**

```
1  avanzar 4 m
2  girar 90 grd
3  avanzar 10 m
4  girar -180 grd
5  avanzar 20 m
```

Figura 18. Evidencia prueba 5 información cargada simular\_comandos

- **Valores ingresados por el usuario:** (100,200)
- **Salida esperada:** (104, 190)
- **Evidencia de resultado:**

```
$simular_comandos 100 200
comando valido :)
La simulación de los comandos, a partir de la posición (100,200), deja al robot en la nueva posición (104,190).
Unidades en metros.
```

Figura 19. Evidencia prueba 5 simular\_comandos

Plan de pruebas: Función simular comandos				
Caso	Información cargada en memoria	Valores de entrada	Resultado esperado	Resultado obtenido
Prueba 1	NO	(0,0)	Mensaje de error	Mensaje de error (La información requerida no está almacenada en memoria)

<b>Prueba 2</b>	SI	(a,b)	Mensaje de error	Mensaje de error (valores ingresados incorrectamente)
<b>Prueba 3</b>	SI	(0,0)	(0.45, 10)	(0.45, 10)
<b>Prueba 4</b>	SI	(-5,-5)	(-1, -15)	(-1, -15)
<b>Prueba 5</b>	SI	(100,200)	(104, 190)	(104, 190)

Figura 20. Tabla plan de pruebas simular\_comandos

## 5.2. Plan de pruebas función en cuadrante:

### Descripción plan de pruebas:

#### 5.2.1. Prueba 1: Elementos no ubicados

En esta prueba intentamos usar la función de simular comandos para comprobar que cuando no hubiera información de los objetos ubicados, lanzara un mensaje de error indicándolo.

- **Valores ingresados por el usuario:** (0, 5, 10, 15)
- **Salida esperada:** Mensaje de error (Los elementos no han sido ubicados todavía (con el comando ubicar\_elementos))
- **Evidencia de resultado:**

```
$en_cuadrante 0 5 10 15
comando valido :)
Los elementos no han sido ubicados todavía (con el comando ubicar_elementos)
```

Figura 21. Evidencia prueba 1 en\_cuadrante

#### 5.2.2. Prueba 2: Valores ingresados de manera incorrecta

En esta prueba intentamos usar la función de simular comandos para comprobar que pasaba si ingresábamos parámetros no esperados, es decir que, a pesar de ya haber ubicado los elementos, al momento de llamar a la función, no ingresamos bien los parámetros, por ejemplo, si ponemos el primero parámetro mayor al segundo parámetro de X

- **Valores ingresados por el usuario:** (1, 0, -5, 1)

- **Salida esperada:** Mensaje de error (La información del cuadrante no corresponde a los datos esperados (x\_min, x\_max, y\_min, y\_max))
- **Evidencia de resultado:**

```
$en_cuadrante 1 0 -5 1
comando valido :)
La información del cuadrante no corresponde a los datos esperados (x_min, x_max, y_min, y_max)
```

Figura 22. Evidencia prueba 2 en\_cuadrante

### 5.2.3. Prueba 3: Valores ingresados de manera incorrecta

En esta prueba intentaremos usar el comando después de haber almacenado en memoria la información necesaria, pero utilizando valores incorrectos como letras en vez de números

- **Valores ingresados por el usuario:** (a, 0, 3, b)
- **Salida esperada:** Mensaje de error (valores ingresados incorrectamente.)
- **Evidencia de resultado:**

```
$en_cuadrante a 0 3 b
comando valido :)
valores ingresados incorrectamente.
```

Figura 23. Evidencia prueba 3 en\_cuadrante

**Información cargada desde memoria para las siguientes pruebas:**

```
1 roca 13 cm 3.45 15.4
2 duna 0.5 km 24.345 67.456
3 crater 0.1 m 11 34
4 roca 43 dm -3.45 -15.4
5 duna 5 km 87 -67.456
6 monticulo 0.55 m -12.2 34.923
7 roca 22 cm 7.45 -19.9
8 duna 65 km 24 90
9 crater 0.55 cm -13.8 76
10 roca 94 dm -99 15.4
11 duna 35 dm 24.345 67.45
12 monticulo 9 m 44 34.923
13 roca 55 dm -100 -28
14 duna 7.5 km 24.345 67
15 monticulo 5 m -80 -52.5
```

Figura 24. Evidencia información cargada en\_cuadrante



#### 5.2.4. Prueba 4: Valores ingresados correctamente cuadrante 1

En esta prueba intentaremos usar el comando ya habiendo almacenado en memoria la información necesaria y probando que aparezcan todos los elementos como resultado del cuadrante 1

- **Valores ingresados por el usuario:** (0, 100, 0, 100)
- **Salida esperada:**
  - roca (3.45, 15.4)
  - duna (24.345, 67.456)
  - duna (24, 90)
  - crater (11, 34)
  - duna (24.345, 67.45)
  - duna (24.345, 67)
  - monticulo (44, 34.923)
- **Evidencia de resultado:**

```
$en_cuadrante 0 100 0 100
comando valido :)
Los elementos ubicados en el cuadrante solicitado son:
- roca (3.45, 15.4)
- duna (24.345, 67.456)
- duna (24, 90)
- crater (11, 34)
- duna (24.345, 67.45)
- duna (24.345, 67)
- monticulo (44, 34.923)
```

Figura 25. Evidencia prueba 4 en\_cuadrante

#### 5.2.5. Prueba 5: Valores ingresados correctamente cuadrante 2

Ya habiendo almacenado en memoria los resultados, queremos comprobar que los elementos del cuadrante 2 aparezcan correctamente en memoria

- **Valores ingresados por el usuario:** (-100, 0, 0, 100)
- **Salida esperada:**
  - monticulo (-12.2, 34.923)
  - crater (-13.8, 76)

- roca (-99, 15.4)
- **Evidencia de resultado:**

```
$en_cuadrante -100 0 0 100
comando valido :)
Los elementos ubicados en el cuadrante solicitado son:
- monticulo (-12.2, 34.923)
- crater (-13.8, 76)
- roca (-99, 15.4)
```

Figura 26. Evidencia prueba 5 en\_cuadrante

### 5.2.6. Prueba 6: Valores ingresados correctamente cuadrante 3

Usaremos el comando para comprobar que una vez almacenada la información en memoria, aparecen los elementos correctos del cuadrante 3

- **Valores ingresados por el usuario:** (-100, 0, -100, 0)
- **Salida esperada:**
  - roca (-3.45, -15.4)
  - roca (-100, -28)
  - monticulo (-80, -52.5)
- **Evidencia de resultado:**

```
$en_cuadrante -100 0 -100 0
comando valido :)
Los elementos ubicados en el cuadrante solicitado son:
- roca (-3.45, -15.4)
- roca (-100, -28)
- monticulo (-80, -52.5)
```

Figura 27. Evidencia prueba 6 en\_cuadrante

### 5.2.7. Prueba 7: Valores ingresados correctamente cuadrante 4

Comprobaremos al igual que con las anteriores pruebas que ya estando la información guardada en memoria, aparecen todos los elementos del cuadrante 4

- **Valores ingresados por el usuario:** (0, 100, -100, 0)
- **Salida esperada:**
  - duna (87, -67.456)

- roca (7.45, -19.9)

- **Evidencia de resultado:**

```
$en_cuadrante 0 100 -100 0
comando valido :)
Los elementos ubicados en el cuadrante solicitado son:
- duna (87, -67.456)
- roca (7.45, -19.9)
```

Figura 28. Evidencia prueba 7 en\_cuadrante

Plan de pruebas: Función en cuadrante				
Caso	Elementos ubicados	Valores de entrada	Resultado esperado	Resultado obtenido
Prueba 1	NO	(0, 5, 10, 15)	Mensaje de error	Mensaje de error (Los elementos no han sido ubicados todavía (con el comando ubicar_elementos))
Prueba 2	SI	(1, 0, -5, 1)	Mensaje de error	Mensaje de error (La información del cuadrante no corresponde a los datos esperados (x_min, x_max, y_min, y_max))
Prueba 3	SI	(a, 0, 3, b)	Mensaje de error	Mensaje de error (valores ingresados incorrectamente.)

<b>Prueba 4</b>	SI	(0, 100, 0, 100)	- roca (3.45, 15.4) - duna (24.345, 67.456) - duna (24, 90) - crater (11, 34) - duna (24.345, 67.45) - duna (24.345, 67) - monticulo (44, 34.923)	- roca (3.45, 15.4) - duna (24.345, 67.456) - duna (24, 90) - crater (11, 34) - duna (24.345, 67.45) - duna (24.345, 67) - monticulo (44, 34.923)
<b>Prueba 5</b>	SI	(-100, 0, 0, 100)	- monticulo (-12.2, 34.923) - crater (-13.8, 76) - roca (-99, 15.4)	- monticulo (-12.2, 34.923) - crater (-13.8, 76) - roca (-99, 15.4)
<b>Prueba 6</b>	SI	(-100, 0, -100, 0)	- roca (-3.45, -15.4) - roca (-100, -28) - monticulo (-80, -52.5)	- roca (-3.45, -15.4) - roca (-100, -28) - monticulo (-80, -52.5)
<b>Prueba 7</b>	SI	(0, 100, -100, 0)	- duna (87, -67.456) - roca (7.45, -19.9)	- duna (87, -67.456) - roca (7.45, -19.9)

Figura 29. Tabla plan de pruebas en\_cuadrante