Proyecto Estructuras de Datos
Entrega 2: Semana 12
Documento de diseño
Integrantes: Luisa Vargas, Manuela García y Liseth Lozano.

Acta de evaluación de la entrega anterior

- Implementar clases
-Cambiar los diagramas de las funciones
-Implementar compilación separada
-Cambiar las funciones cargar_comandos y simular_comandos

TADs

TDA Robot:

Datos Mínimos:

-coordenada X: valor flotante que corresponde a la posición en el eje x del robot.
-coordenada Y: valor flotante que corresponde a la posición en el eje y del robot.
-orientación: valor flotante que corresponde hacia el grado en el plano que está mirando el robot.

Operaciones:

-ObtenerCoordenadaX:Retorna el valor actual de la coordenada en X.
-ObtenerCoordenadaY:Retorna el valor actual de la coordenada en Y.
-ObtenerOrientación:Retorna el valor actual hacia cuantos grados está apuntado el robot.
-FijarCoordenadaX(CoordenadaX):Fija el nuevo valor para la coordenada X.
-FijarCoordenadaY(CoordenadaY):Fija el nuevo valor para la coordenada Y.
-FijarOrientación(Orientación):Fija el nuevo valor para la Orientación..
-SimularComandos(CoordenadaX,CoordenadaY):Permite simular el resultado de los comandos de movimiento que se enviarán al robot.
-GuardarComandos(tipo_archivo,nombre_archivo):Guarda en el archivo nombre_archivo la información solicitada de acuerdo al tipo de archivo: comandos almacena en el archivo la información de comandos de movimiento y de análisis que debe ejecutar el robot.

TDA Comando:

Datos Mínimos:

-nombre_comando: Nombre del comando del robot.
-descripción:Descripción de lo que hace ese comando en específico.

Operaciones:

-Obtenernombre_Comando:Retorna el nombre actual del comando.
-Obtenerdescripción:Retorna la descripción del comando
-Fijarnombre_Comando(nombre_Comando):Fija el nuevo nombre del comando
-Fijardescripción(descripción):Fija la nueva descripción del comando
-mostarComandos:imprime todos los comandos del robot.

TDA Análisis:

Datos Mínimos:

-tipo_analisis:Especifica el tipo de análisis que puede ser fotografiar, composición o perforar.
-objeto:nombre del elemento que se quiere analizar.
-comentario:agrega más información del elemento o análisis.

Operaciones:

-Obtenertipo_analisis:Retorna el tipo de análisis actual.
-Obtenerobjeto:Retorna el objeto actual del análisis.
-Obtenercomentario:Retorna el comentario actual del análisis.
-Fijartipo_analisis(tipo_analisis):Fija el nuevo tipo de análisis.
-Fijarobjeto(objeto):Fija el nuevo objeto del análisis.
-Fijarcomentario(comentario):Fija el nuevo comentario del análisis.
-agregar_analisis(tipo_analisis, objeto, comentario):Agrega el comando de análisis descrito a la lista de comandos del robot.
-cargar_comandos(nombre_archivo):Carga al vector de análisis la lista de comandos tipo análisis encontrados en el archivo.

TDA Elemento:

Datos Mínimos:

-tipo_comp:Componente del elemento que puede ser uno entre roca, cráter, montículo o duna.
-unidad_med:valor con la convención que se usó para su medición (centímetros, metros, ...).
-tamaño:valor real que da cuenta de qué tan grande es el elemento
-coordX:Número real que da información de la posición en X del elemento en el eje coordenado.
-coordY:Número real que da información de la posición en Y del elemento en el eje coordenado.

Operaciones:

-Obtenertipo_comp:Retorna el tipo de componente actual.
-Obtenerunidad_med:Retorna la unidad de medida actual..
-Obtenertamaño:Retorna el tamaño del componente.
-ObtenercoordX:Retorna la posición actual en el componente X.
-ObtenerrcoordY:Retorna la posición actual en el componente Y.
-Fijartipo_comp(tipo_comp):Fija el tipo de componente actual.
-Fijarunidad_med(unidad_med):Fija la unidad de medida..
-Fijartamaño(tamaño):Fija el tamaño del componente.
-FijarcoordX(coordX):Fija la coordenada en X.
-FijarcoordY(coordY):Fija la coordenada en Y..
-agregar_elemento(tipo_comp, tamaño, unidad_med, coordX, coordY):Agrega el componente o elemento descrito a la lista de puntos de interés del robot
-cargar_elementos(nombre_archivo):Carga en memoria los datos de puntos de interés o elementos contenidos en el archivo.

TDA Movimiento:

Datos Mínimos:

-tipo_movimiento: Corresponde al tipo de movimiento puede ser de dos tipos: avanzar o girar.

-magnitud:corresponde al valor del movimiento.

-unidad_med: corresponde a la convención con la que se mide la magnitud del movimiento.

Operaciones:

-Obtenertipo_movimiento:Retorna el tipo de movimiento actual.

-Obtenermagnitud:Retorna la magnitud actual.

-Obtenerunidad_med:Retorna la unidad de medida actual.

-Fijartipo_movimiento(tipo_movimiento):Fija el nuevo tipo de movimiento.

-Fijarmagnitud(magnitud):Fija la nueva magnitud del movimiento.

-Fijarunidad_med(unidad_med):Fija la unidad de medida del movimiento.

-agregar_movimiento(tipo_mov, magnitud, unidad_med):Agrega el comando de movimiento descrito a la lista de comandos del robot.

-cargar_comandos(nombre_archivo):Carga al vector de movimientos la lista de comandos tipo movimiento encontrados en el archivo.

TDA Quadtree:

Datos Mínimos:

- root : Node*: Puntero al nodo raíz del quadtree.

- capacity int : Capacidad máxima de elementos que puede contener cada nodo antes de subdividirse.

- x_min float:Límite del espacio que abarca el quadtree en las coordenada x mínima.

- x_max float:Límite del espacio que abarca el quadtree en las coordenada x máxima.

- y_min float:Límite del espacio que abarca el quadtree en las coordenada y mínima

- y_max float: Límite del espacio que abarca el quadtree en las coordenada y máxima.

Operaciones:

+ Quadtree(int capacity, float x_min, float x_max, float y_min, float y_max) : constructor : Constructor que inicializa el quadtree con la capacidad máxima de elementos por nodo y los límites del espacio.

+ ~Quadtree() : destructor : Destructor que libera la memoria asociada con el quadtree.

+ insertarElemento(const Elemento& elemento) : void: Método público para insertar un elemento en el quadtree.

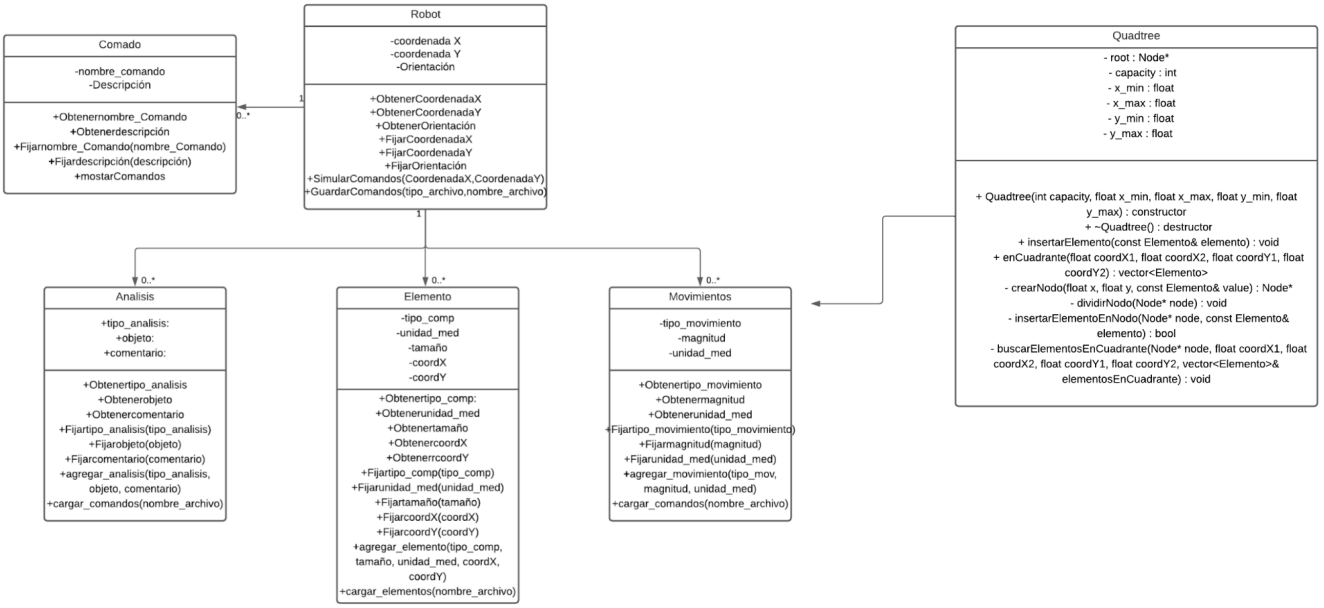
+enCuadrante(float coordX1, float coordX2, float coordY1, float coordY2) : vector<Elemento>: Método público para buscar elementos en un cuadrante específico, definido por las coordenadas x e y.

- crearNodo(float x, float y, const Elemento& value) : Node* :Crea un nuevo nodo con las coordenadas x e y y un valor asociado.

- dividirNodo(Node* node) : void :Crea un nuevo nodo con las coordenadas x e y y un valor asociado.: Divide un nodo en 4 nodos hijos cuando se alcanza la capacidad máxima de elementos en ese nodo.

- insertarElementoEnNodo(Node* node, const Elemento& elemento) : bool :Inserta un elemento en un nodo específico del quadtree. Si el nodo alcanza su capacidad máxima, se divide.

-buscarElementosEnCuadrante(Node* node, float coordX1, float coordX2, float coordY1, float coordY2, vector<Elemento>& elementosEnCuadrante) : void : Método privado que busca elementos en un cuadrante específico a partir de un nodo dado, almacenando los elementos encontrados en un vector.



Clases implementadas

Para el proyecto se declararon las siguientes **clases:**

//**Clase Elemento**

```
class Elemento {
private:
  //Atributos de la clase Elemento
  string tipo_comp;
  string unidad_med;
  float tamano;
  float coordX;
  float coordY;

public:
  // Constructores de la clase Elemento
```

```
  Elemento() {}
    Elemento(string tipo, string unidad, float tam, float x, float y) : tipo_comp(tipo), unidad_med(unidad), tamano(tam), coordX(x), coordY(y) {}

  //Getters de la clase Elemento
  string getTipoComp() const {
   return tipo_comp;
  }

  string getUnidadMed() const {
   return unidad_med;
  }

  float getTamano() const {
   return tamano;
  }

  float getCoordX() const {
   return coordX;
  }

  float getCoordY() const {
   return coordY;
  }
};

//Clase Quadtree
class Quadtree {
private:
  // Clase interna para representar un nodo
  class Node {
    public:
      float x, y;   // Coordenadas del nodo
      string tipo_comp;    // Valor del nodo
      Node *nw, *ne, *sw, *se;  // Punteros a los cuatro hijos del nodo

      //Constructor de la clase nodo
       Node(float x, float y, string tipo_comp) : x(x), y(y), tipo_comp(tipo_comp), nw(nullptr), ne(nullptr), sw(nullptr), se(nullptr) {}
  };

  Node *root;  // Puntero al nodo raíz

public:
```

```
Quadtree() : root(nullptr) {} //Constructor que inicializa al Quadtree como vacío
 ~Quadtree() { delete root; }  //Destructor para liberar la memoria de todos los nodos del árbol

// Funcion para insertar nodos
void insert(float x, float y, string tipo_comp) { //Se inserta un nodo con las coordenadas del elemento y tipo_comp del elemento
  if (root == nullptr) { //Si el árbol está vacío, se crea un nuevo nodo con los parámetros de la función
   root = new Node(x, y, tipo_comp);
   return;
  }

  Node *actual = root; //Se declara un nodo que apunta a la raíz del Quadtree
  while (true) { //Ciclo while indefinido
    if (x < actual->x && y < actual->y) { //Se comparan las coordenadas recibidas con las del nodo actual
      //Si x,y son menores que las coordenadas del nodo actual, significa que deben ir en el cuadrante noroeste (nw) del nodo actual
     if (actual->nw == nullptr) { //Si el cuadrante noroeste está vacío se crea un nuevo nodo en esa posición con los parámetros de la función
      actual->nw = new Node(x, y, tipo_comp);
      return;
     }
      actual = actual->nw; // Si el cuadrante no está vacío, se cambia el valor de actual al nodo nw
    }
    else if (x >= actual->x && y < actual->y) { //Si x es mayor o igual que el x del nodo actual, y y es menor que el y del nodo actual, significa que deben ir en el cuadrante noreste (ne) del nodo actual
      if (actual->ne == nullptr) { //Si el cuadrante noreste está vacío se crea un nuevo nodo en esa posición con los parámetros de la función
       actual->ne = new Node(x, y, tipo_comp);
       return;
      }
       actual = actual->ne; // Si el cuadrante no está vacío, se cambia el valor de actual al nodo ne
    }
    else if (x < actual->x && y >= actual->y) { //Si x es menor que el x del nodo actual, y y es mayor o igual que el y del nodo actual, significa que deben ir en el cuadrante suroreste (sw) del nodo actual
     if (actual->sw == nullptr) { //Si el cuadrante suroreste está vacío se crea un nuevo nodo en esa posición con los parámetros de la función
       actual->sw = new Node(x, y, tipo_comp);
```

```
            return;
        }
            actual = actual->sw; // Si el cuadrante no está vacío, se cambia el valor de actual al nodo sw
       }
       else {
        if (actual->se == nullptr) { //Si el cuadrante sureste está vacío se crea un nuevo nodo en esa posición con los parámetros de la función
            actual->se = new Node(x, y, tipo_comp);
            return;
        }
            actual = actual->se; // Si el cuadrante no está vacío, se cambia el valor de actual al nodo se
       }
     }
   }

   //Esta funcion sirve para imprimir todos los nodos del Quadtree
   void print() {
      if (root != nullptr) { //Si el árbol no está vacío, se imprime el árbol y sus subárboles
         printNode(root);
      }
   }

     //Esta función determina si los elementos cargados se encuentran dentro de un cuadrante especifico
       vector<Elemento> enCuadrante(float coordX1, float coordX2, float coordY1, float coordY2) {
     if (coordX1 >= coordX2 || coordY1 >= coordY2) { //Verifica que los datos sean validos
         cout << "(Formato erróneo) La información del cuadrante no corresponde a los datos esperados (x_min, x_max, y_min, y_max)." << endl;
       return {};
     }

      if (root == nullptr) { //Verifica si el árbol está vacío, e imprime un mensaje de error si es así
          cout << "(No hay información) Los elementos no han sido ubicados todavía (con el comando ubicar_elementos)." << endl;
       return {};
     }

       vector<Elemento> elementosEnCuadrante; //Se inicializa una lista vacía de elementos para almacenar los que están en el cuadrante delimitado
```

```
            buscarElementosEnCuadrante(root, coordX1, coordX2, coordY1, coordY2, elementosEnCuadrante); //Se llama la función buscarElementosEnCuadrante

     if(!elementosEnCuadrante.empty()) { //Si hay elementos dentro del cuadrante
       cout << "(Resultado exitoso) Los elementos ubicados en el cuadrante solicitado son:" << endl;
       for (const Elemento& elemento : elementosEnCuadrante) { //Impresión de los elementos en el cuadrante
                cout << "Tipo componente: " << elemento.getTipoComp() << ", x: " << elemento.getCoordX() << ", y: " << elemento.getCoordY() << endl;
       }
     }
     else { //Impresión de que no se encontraron elementos
        cout << "(Resultado exitoso) No se encontraron elementos en el cuadrante solicitado." << endl;
     }
   return elementosEnCuadrante;
}

private:
   // Función recursiva para imprimir un nodo y sus hijos
   void printNode(Node *node) {
          //Se imprime en la consola las coordenadas x e y del nodo, junto con el tipo de componente que contiene
          cout << "x: " << node->x << ", y: " << node->y << ", Tipo componente: " << node->tipo_comp << endl;

           //Se verifica si el nodo tiene hijos en los cuadrantes, y si es así se imprime recursivamente
       if (node->nw != nullptr) {
          printNode(node->nw);
       }
       if (node->ne != nullptr) {
          printNode(node->ne);
       }
       if (node->sw != nullptr) {
          printNode(node->sw);
       }
       if (node->se != nullptr) {
          printNode(node->se);
       }
   }

  //Esta función busca los elementos que se encuentran dentro de un cuadrante en específico
```

```
    void buscarElementosEnCuadrante(Node *node, float coordX1, float coordX2, float coordY1, float coordY2, vector<Elemento> &elementosEnCuadrante) {
     if (node == nullptr) return; //Se verifica si el nodo está vacío, y si es así se sale de la función

   //Se verifica si el nodo actual está dentro del cuadrante buscado,si es así, se añade un nuevo objeto Elemento al vector elementosEnCuadrante con los datos del nodo actual.
    if (node->x >= coordX1 && node->x <= coordX2 && node->y >= coordY1 && node->y <= coordY2) {
                elementosEnCuadrante.push_back(Elemento(node->tipo_comp, "", 0, node->x, node->y));
    }

    if (node->x > coordX1) { //Llamada recursiva a los hijos del nodo actual que están dentro del cuadrante buscado
       //Si el nodo está a la izquierda del cuadrante, se buscan sus hijos en la parte izquierda del Quadtree
              buscarElementosEnCuadrante(node->nw, coordX1, coordX2, coordY1, coordY2, elementosEnCuadrante);
              buscarElementosEnCuadrante(node->sw, coordX1, coordX2, coordY1, coordY2, elementosEnCuadrante);
    }

    if (node->x <= coordX2) { //Llamada recursiva a los hijos del nodo actual que están dentro del cuadrante buscado
        //Si el nodo está a la derecha del cuadrante, se buscan sus hijos en la parte derecha del Quadtree
              buscarElementosEnCuadrante(node->ne, coordX1, coordX2, coordY1, coordY2, elementosEnCuadrante);
              buscarElementosEnCuadrante(node->se, coordX1, coordX2, coordY1, coordY2, elementosEnCuadrante);
    }
  }
};

//Clase Robot
class Robot {
private:
  float coordenadaX;
  float coordenadaY;
  float orientacion;

public:
```

```
  Robot() : coordenadaX(0), coordenadaY(0), orientacion(90) {} // Constructor de la clase Robot
};

//Clase Comando
class Comando {
private:
  string nombre_comando;
  string descripcion;

public:
  Comando(string nombre, string desc) : nombre_comando(nombre), descripcion(desc) {} // Constructor de la clase Comando

  //Getters de la clase Comando
  string getNombreComando() const {
    return nombre_comando;
  }

  string getDescripcion() const {
    return descripcion;
  }
};

//Clase Analisis
class Analisis {
private:
    string tipo_analisis;
    string objeto;
    string comentario;

public:
    Analisis() {} //Constructor de la clase Analisis
        Analisis(string tipo, string obj, string com) : tipo_analisis(tipo), objeto(obj), comentario(com) {} // Constructor de la clase Analisis

    //Getters de la clase Analisis
    string getTipoAnalisis() const {
      return tipo_analisis;
    }

   string getObjeto() const{
     return objeto;
   }
```

```
  string getComentario() const {
    return comentario;
  }
};

//Clase Movimiento
class Movimiento {
private:
  string tipo_movimiento;
  float magnitud;
  string unidad_med;

public:
  Movimiento() {} //Constructor de la clase Movimiento
   Movimiento(string tipo, float mag, string unidad) : tipo_movimiento(tipo), magnitud(mag),
unidad_med(unidad) {} // Constructor de la clase Movimiento

  //Getters de la clase Movimiento
  string getTipoMovimiento() const {
    return tipo_movimiento;
  }

  float getMagnitud() const {
    return magnitud;
  }

  string getUnidadMed() const {
    return unidad_med;
  }
};
```

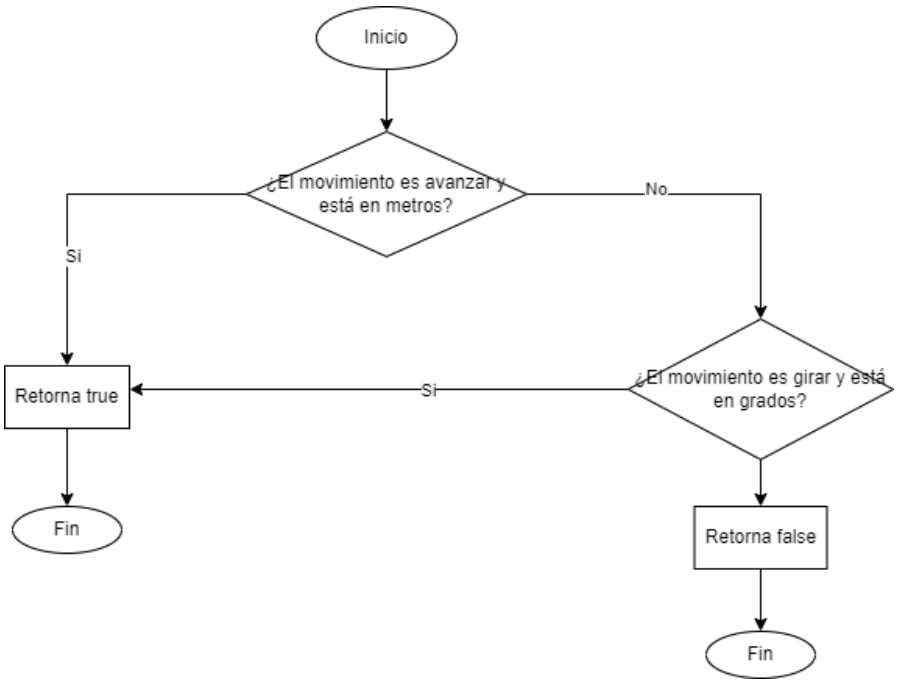
Funciones Implementadas

esUnidadValidaParaMovimiento

En esta funciòn se pasan por paràmetro dos strings (tipo_movimiento, string unidad_med) y se verifican las unidades de medida de la siguiente forma:

→Si es avanzar, se verifica que sean metros

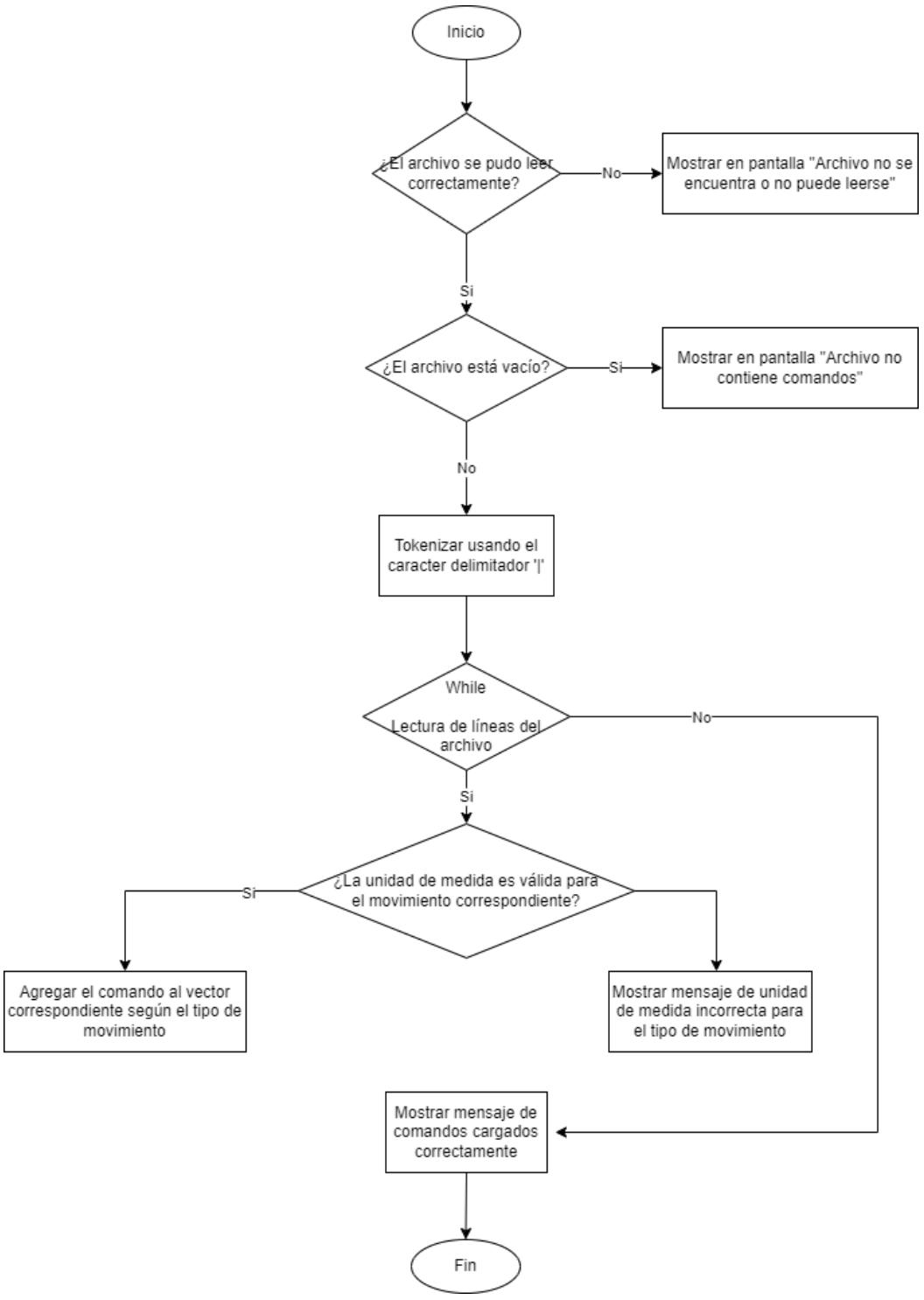
→Si es girar, se verifica que sean grados



cargarComandos

En esta función se pasan por parámetro los vectores de movimientos y análisis, junto con el nombre del archivo que se debe leer.

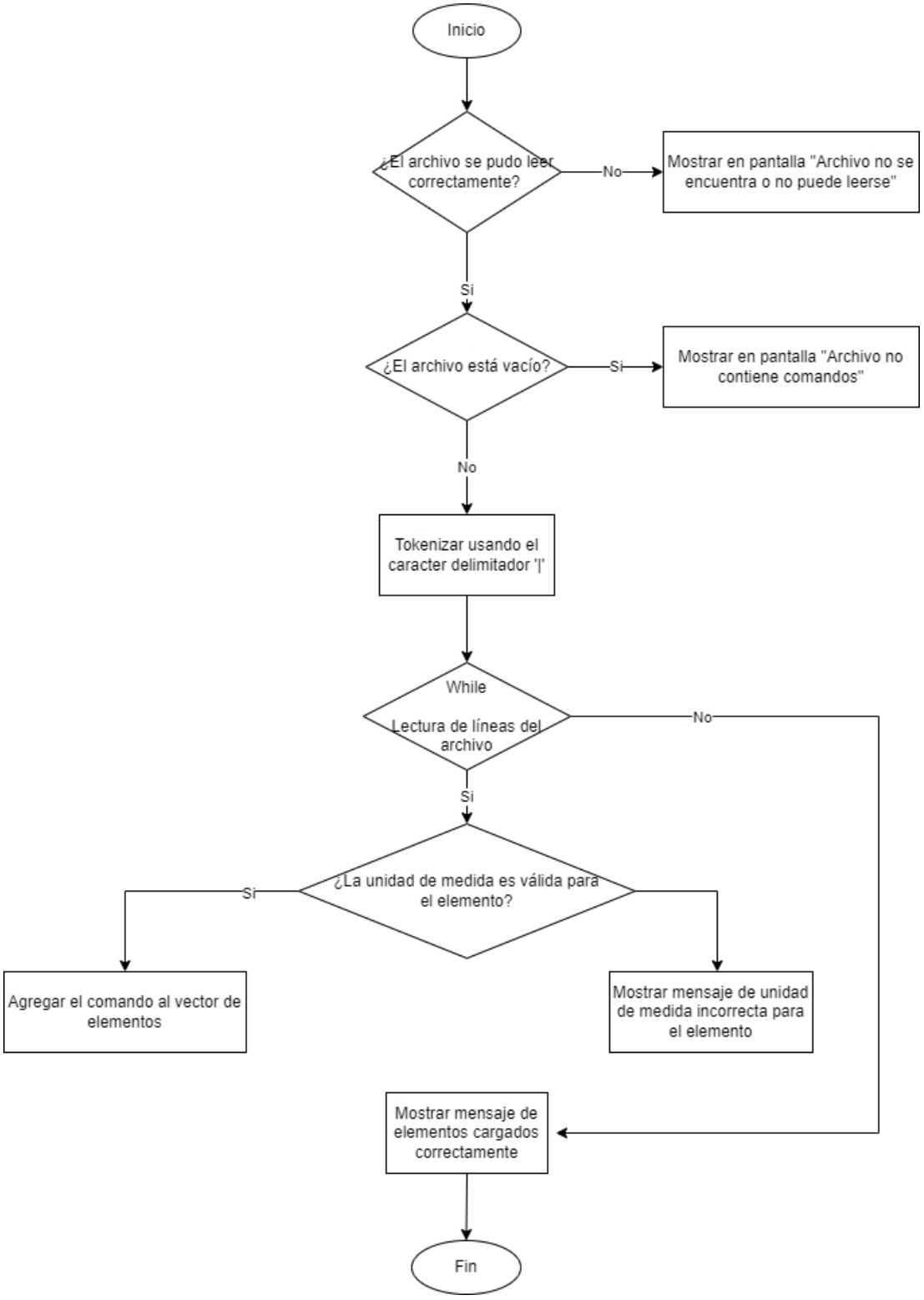
Se realiza la lectura y apertura del archivo y si todo sale bien, se procede a tokenizar cada línea del archivo usando el caracter delimitador '|' y la función getline, verificando también las unidades de medida (metros o grados dependiendo del desplazamiento) a través de la función **esUnidadValidaParaMovimiento** y posteriormente, se agrega al respectivo vector.



cargarElementos

En esta función se pasan por parámetro el vector de elementos, junto con el nombre del archivo que se debe leer.

Se realiza la lectura y apertura del archivo y si todo sale bien, se procede a tokenizar cada línea del archivo usando el caracter delimitador '|' y la función find, verificando también la unidad de medida del elemento (metros) y posteriormente, se agrega al respectivo vector.



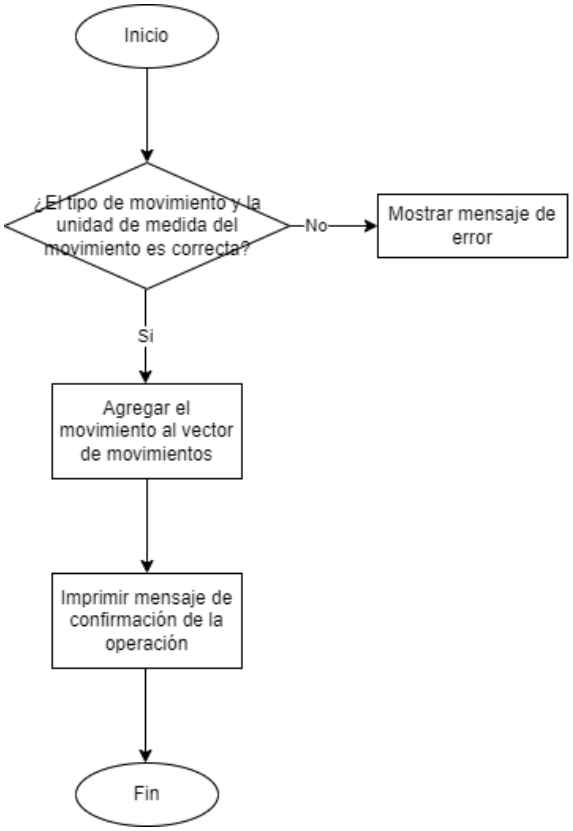
agregar_movimiento

Esta función agrega un nuevo movimiento a un vector de movimientos, donde cada movimiento contiene tres elementos: tipo de movimiento, magnitud y unidad de medida.

Los cuatro parámetros de la función son:

- movimiento: un vector de movimientos existente, al que se agrega el nuevo movimiento.
- tipo_mov: una cadena de caracteres que indica el tipo de movimiento que se va a realizar. Solo se permiten los tipos de movimiento "avanzar" y "girar".
- magnitud: un número decimal que representa la magnitud del movimiento.
- unidad_med: una cadena de caracteres que indica la unidad de medida de la magnitud del movimiento solo se permite metros para el caso de avanzar y grados para el caso de girar.

Antes de agregar el nuevo movimiento al vector de movimientos, la función verifica si la información del movimiento es adecuada. Si el tipo_mov es "avanzar" o "girar" y si es metros o grados, entonces la función creará un nuevo movimiento y lo agrega al final del vector de movimientos existente.
Si la información del movimiento no es adecuada, la función imprimirá un mensaje de error y no se agregará ningún movimiento al vector de movimientos.
En cualquier caso, la función imprimirá un mensaje indicando si el comando de movimiento ha sido agregado exitosamente o no.

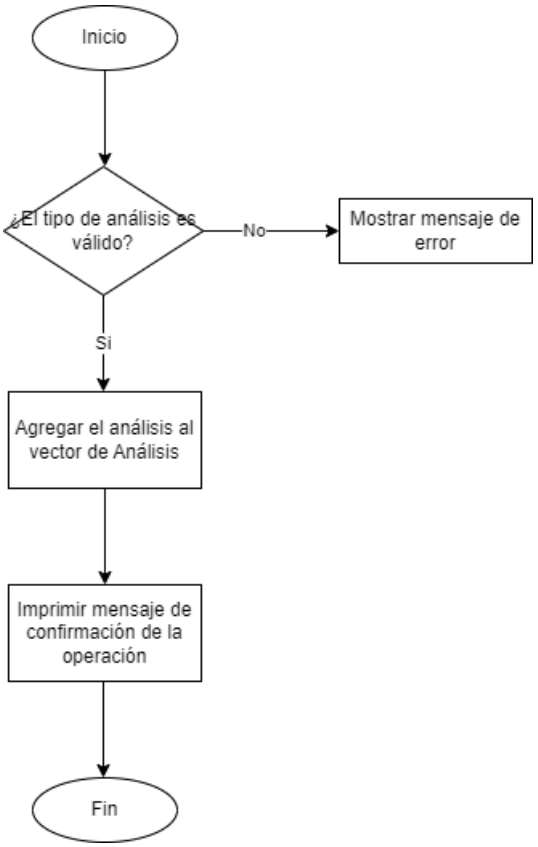


agregar_analisis:
Esta función agrega un nuevo análisis a un vector de análisis, donde cada análisis contiene tres elementos: tipo de análisis, objeto y comentario.

Los tres parámetros de la función son:

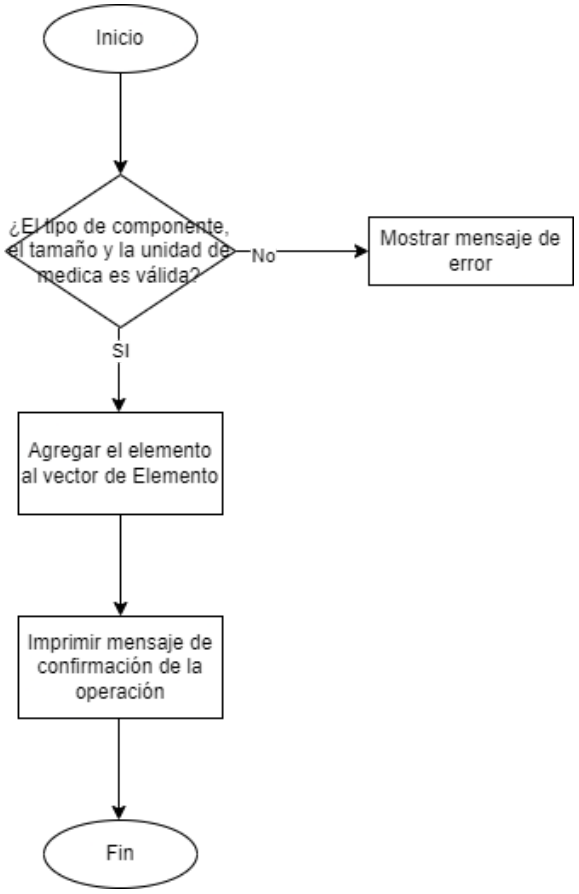
- tipo_analisis: una cadena de caracteres que indica el tipo de análisis que se va a realizar. Solo se permiten los tipos de análisis "fotografiar", "composicion" y "perforar".
- objeto: una cadena de caracteres que representa el objeto que se va a analizar.
- comentario: una cadena de caracteres que contiene algún comentario o descripción adicional sobre el análisis.

El último parámetro es un vector de análisis existente, al que se agrega el nuevo análisis.
Antes de agregar el nuevo análisis al vector de análisis, la función verifica si el tipo_analisis es válido, es decir, si es uno de los tipos de análisis permitidos. Si el tipo_analisis no es válido o el objeto está vacío, la función imprimirá un mensaje de error y saldrá de la función sin agregar el nuevo análisis.
Si el tipo_analisis es válido y el objeto no está vacío, la función creará un nuevo análisis y lo agregará al final del vector de análisis existente. Luego, la función imprimirá un mensaje de éxito.



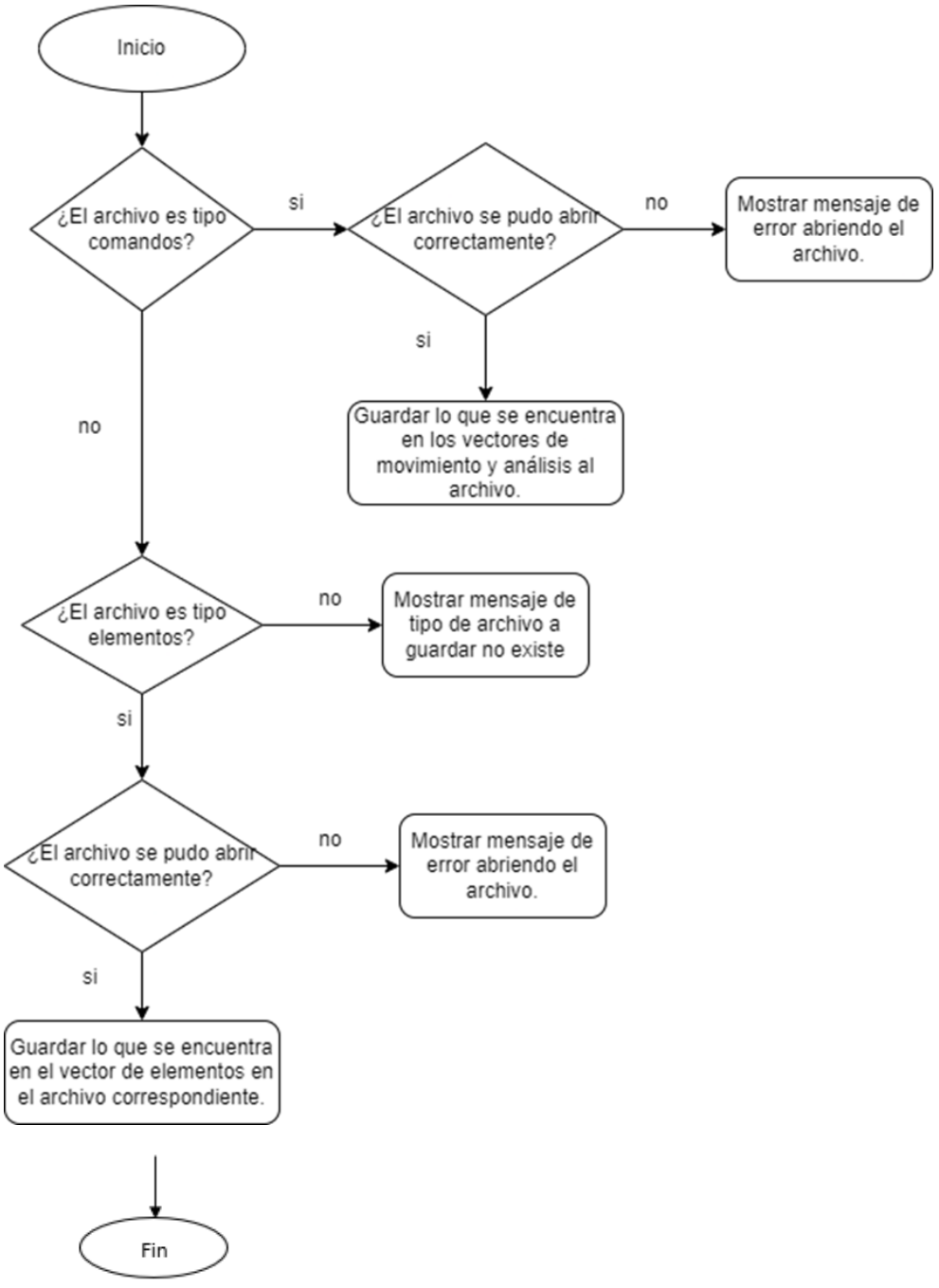
agregarElemento:
La función "agregarElemento" recibe como parámetros información sobre un elemento, como su tipo de componente, tamaño, unidad de medida y coordenadas de ubicación (X y Y), así como también un vector de elementos. La función verifica que la información ingresada sea adecuada y, si es así, crea una nueva instancia de estructura de Elementos con la información

ingresada y la agrega al final del vector de elementos. Si la información no es adecuada, la función simplemente emite un mensaje de error indicando que la información ingresada no corresponde a los datos pedidos. En general, la función se encarga de agregar un nuevo elemento al vector de elementos si se cumplen ciertas condiciones y emite un mensaje de éxito o fracaso dependiendo del caso.



guardar:

La función guardar pasa por parámetro el nombre del archivo, tipo de archivo (comandos o elementos) y los vectores de elementos, movimientos y análisis. Se realiza una verificación de que tipo de archivo se debe abrir, para hacer una apertura del mismo y si todo abre de forma correcta se procede a guardar todos los elementos del correspondiente vector(es) en el archivo, para finalmente cerrarlo.



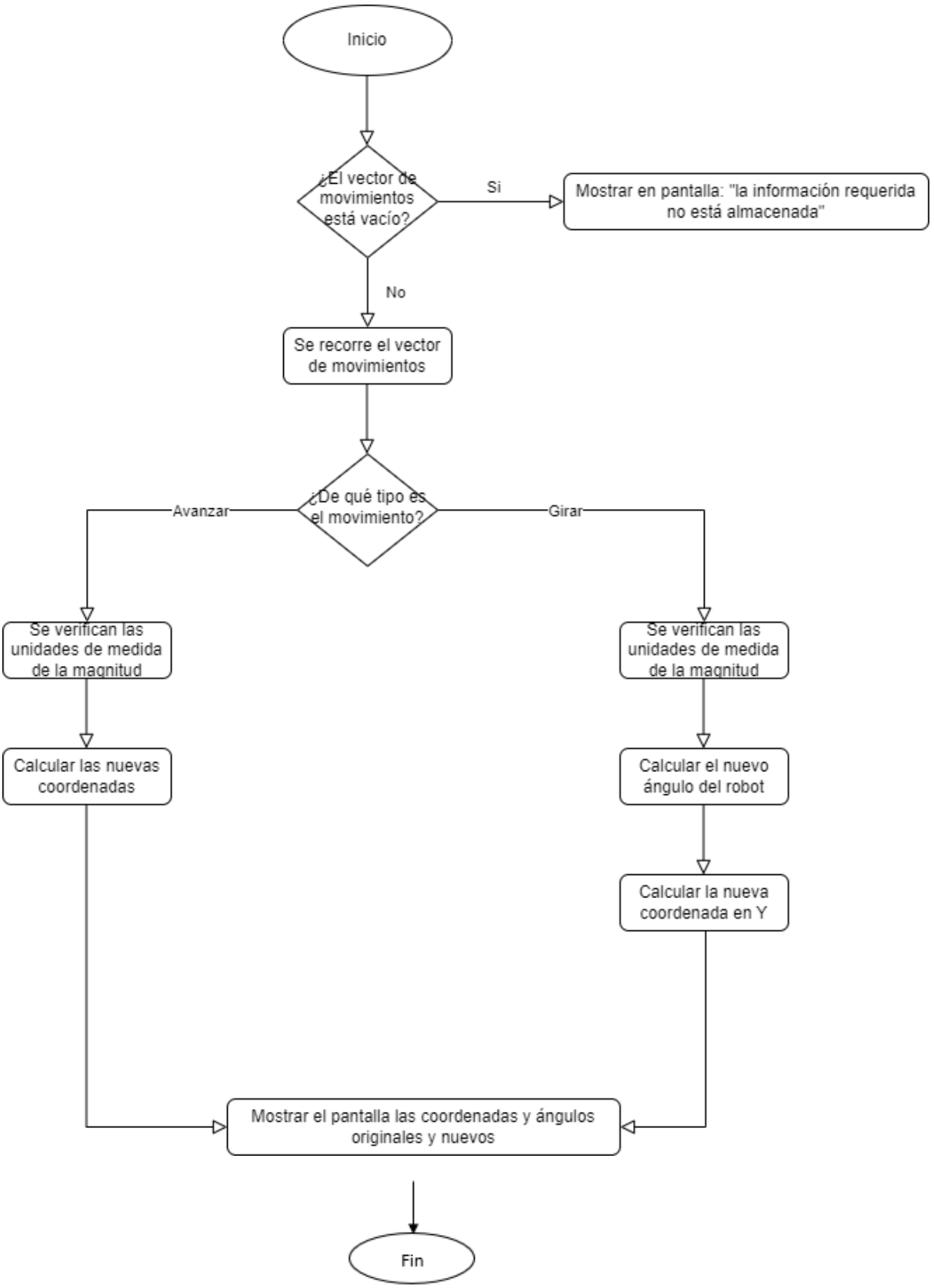
simularComandos

La función simular_comando recibe como parámetros tres variables: coordenadaX y coordenadaY, que representan la posición inicial del robot, y movimiento, que es un vector de

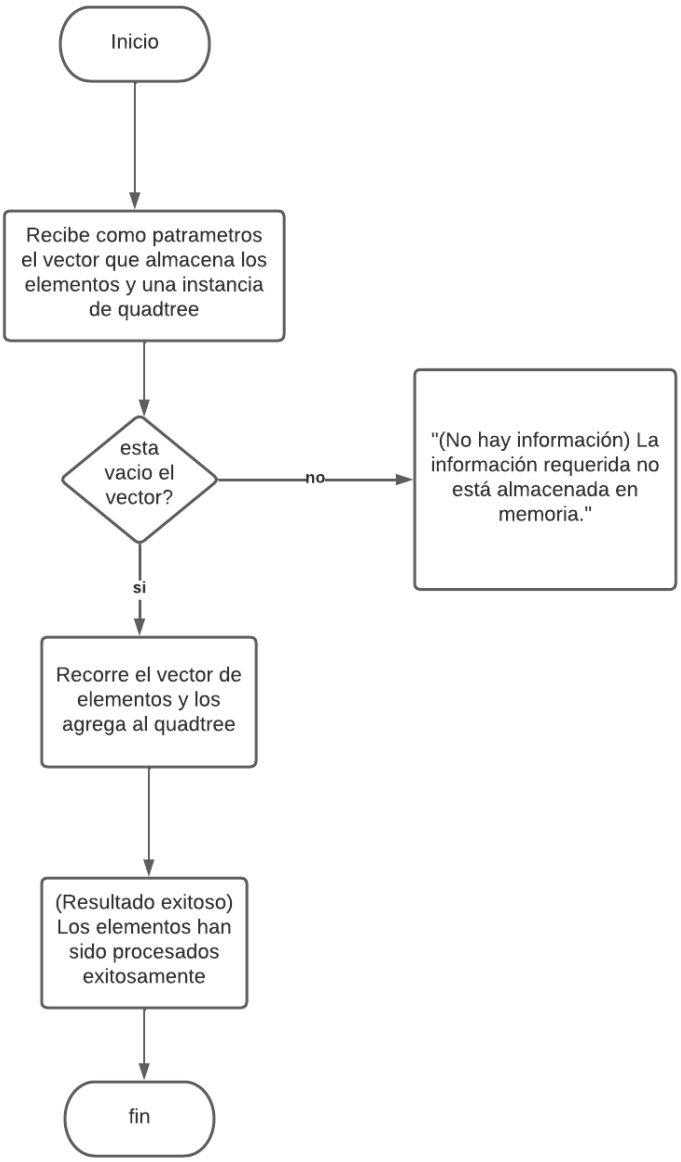
la estructura Movimiento que contiene información sobre los movimientos que el robot debe realizar.

El ciclo for recorre cada elemento del vector movimiento, accediendo a cada elemento mediante el iterador it, dependiendo del tipo de movimiento se verifican las unidades de medida del movimiento.

Si el tipo_movimiento_actual es "avanzar" y la unidad_med_actual es "metros", se calcula la distancia que se debe recorrer y se actualiza la posición del robot en función de la distancia recorrida y el ángulo actual. Mientras que, si el tipo_movimiento_actual es "girar" y la unidad_med_actual es "grados", se calcula el ángulo del giro y se actualiza la posición del robot en función del ángulo y la coordenada en Y.

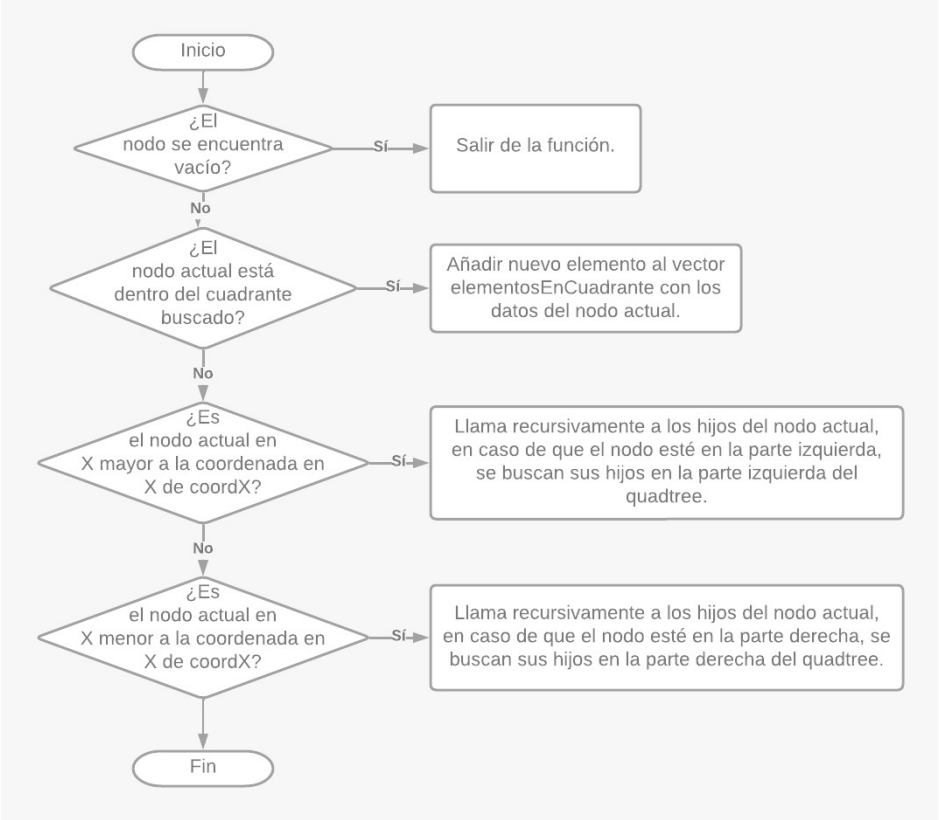

Inicio
¿El vector de movimientos está vacío?
Si
Mostrar en pantalla: "la información requerida no está almacenada"
No
Se recorre el vector de movimientos
¿De qué tipo es el movimiento?
Avanzar
Girar
Se verifican las unidades de medida de la magnitud
Calcular las nuevas coordenadas
Se verifican las unidades de medida de la magnitud
Calcular el nuevo ángulo del robot
Calcular la nueva coordenada en Y
Mostrar el pantalla las coordenadas y ángulos originales y nuevos
Fin

ubicarElementos:
Esta función "ubicarElementos" recibe un vector de objetos Elemento y una instancia del árbol Quadtree. Primero, la función verifica si el vector de elementos está vacío y en caso contrario, agrega cada elemento al Quadtree mediante la función "insert". Después de agregar todos los elementos, la función imprime el árbol Quadtree y muestra un mensaje de éxito en la operación. En resumen, esta función agrega elementos al árbol Quadtree y lo imprime



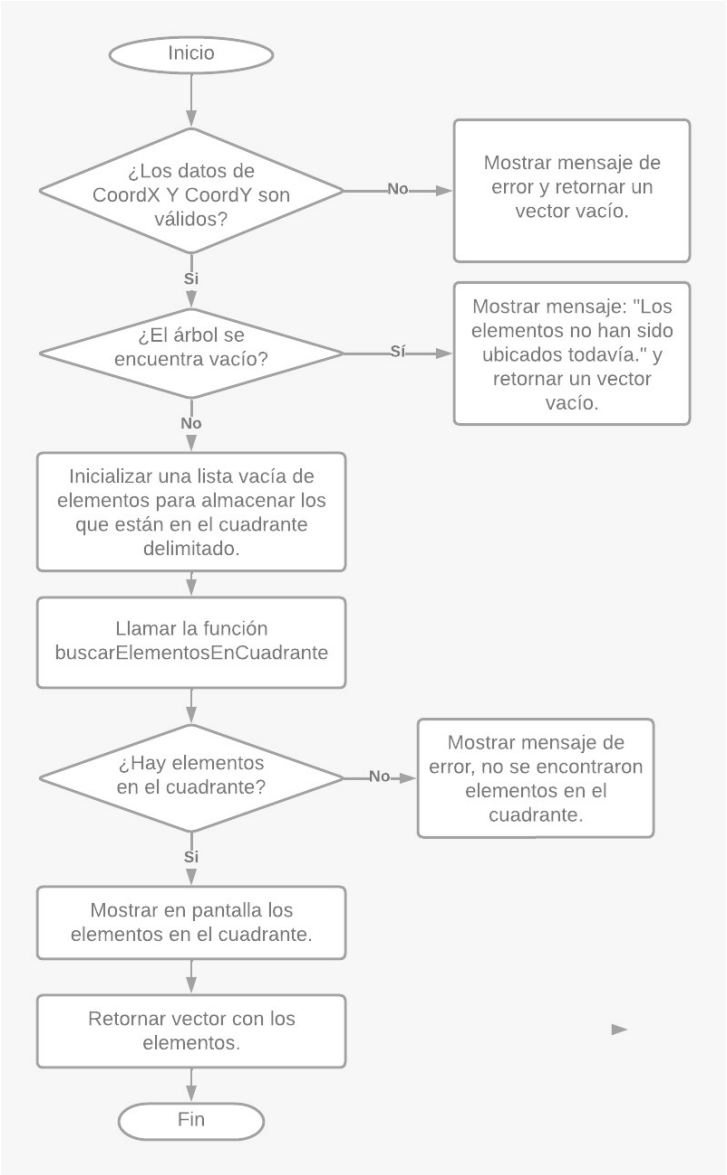
buscarElementosEnCuadrante:
Esta función "buscarElementosEnCuadrante" recibe un nodo del árbol Quadtree, las coordenadas de un cuadrante delimitado por dos puntos (coordX1, coordY1) y (coordX2, coordY2), y un vector de elementos. La función verifica si el nodo está dentro del cuadrante buscado, y si es así, añade un nuevo objeto Elemento al vector. La función también realiza llamadas recursivas a los hijos del nodo actual que están dentro del cuadrante buscado. En

resumen, esta función busca y añade los elementos dentro de un cuadrante específico del Quadtree.



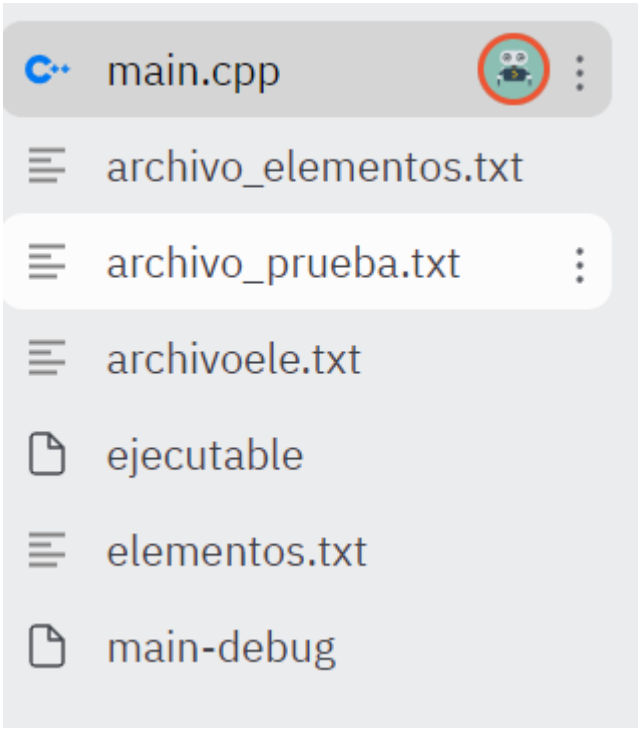
enCuadrante:

La función "enCuadrante" recibe como parámetros las coordenadas de un cuadrante delimitado por dos puntos (coordX1, coordY1) y (coordX2, coordY2). Si los datos son válidos y el árbol de búsqueda no está vacío, la función busca los elementos almacenados en dicho cuadrante y los almacena en un vector. Si se encontraron elementos, los muestra en pantalla junto con su información relevante. En caso contrario, muestra un mensaje indicando que no se encontraron elementos. La función retorna el vector con los elementos encontrados o un vector vacío en caso de errores.



Plan de pruebas:

```
~/ProyectoE2V2Corregido$ g++ -std=c++11 -o ejecutable main.cpp
~/ProyectoE2V2Corregido$
```




```
~/ProyectoE2V2Corregido$ g++ -std=c++11 -o ejecutable main.cpp
~/ProyectoE2V2Corregido$ gdb ejecutable
GNU gdb (GDB) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ejecutable...
(No debugging symbols found in ejecutable)
(gdb) run
Starting program: /home/runner/ProyectoE2V2Corregido/ejecutable
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/nix/store/4nlgxhb09sdr51nc9hdm8az5b08vzkgx-glibc-2.35-163/lib/libthread_db.so.1".
Bienvenido a la interfaz del Curiosity
Ingrese el comando ayuda para ver todos los comandos
$
```

```
Comando: agregar_analisis tipo_analisis objeto comentario
Descripción: Agrega el comando de análisis descrito a la lista de comandos del robot “Curiosity”.
--------------------------------------------------------------------------------------------------------------------
Comando: agregar_elemento tipo_comp tamaño unidad_med coordX coordY
Descripción: Agrega el componente o elemento descrito a la lista de puntos de interés del robot “Curiosity”.
--------------------------------------------------------------------------------------------------------------------
Comando: guardar tipo_archivo nombre_archivo
Descripción: Guarda en el archivo nombre_archivo la información solicitada de acuerdo al tipo de archivo
--------------------------------------------------------------------------------------------------------------------
Comando: simular_comandos coordX coordY
Descripción: Agrega el componente o elemento descrito a la lista de puntos de interés del robot “Curiosity”.
--------------------------------------------------------------------------------------------------------------------
Comando: ubicar_elementos
Descripción: Utiliza la información de puntos de interés almacenada en memoria para ubicarlos en una estructura de
erárquica adecuada, para luego realizar consultas geográficas sobre estos elementos.
--------------------------------------------------------------------------------------------------------------------
Comando: en_cuadrante coordX1 coordX2 coordY1 coordY2
Descripción: Utiliza la estructura creada con el comando anterior para retornar una lista de los componentes o ele
ue están dentro del cuadrante geográfico descrito por los límites de coordenadas en x y y.
--------------------------------------------------------------------------------------------------------------------
Comando: salir
Descripción: Termina la ejecución de la aplicación.
--------------------------------------------------------------------------------------------------------------------
Ingrese el comando ayuda para ver todos los comandos
$exit
Comando inválidoIngrese el comando ayuda para ver todos los comandos
$salir
[Inferior 1 (process 12080) exited normally]
(gdb) exit
~/ProyectoE2V2Corregido$
```