Pontificia Universidad Javeriana
Facultad de Ingeniería
Carrera de ciencia de datos

Diseño de solución
Entrega 02
Proyecto Estructuras de Datos

Autor:
Juan José Álvarez Órtiz
Sebastián Córdoba Valderrama
Samuel Peña García
Mauricio Rodríguez Córdoba
Paula Sofía Torres Rodríguez

ÍNDICE

1. Contexto

El objetivo del presente proyecto es construir un sistema que permita simular las actividades e interacción entre el vehículo “Curiosity” y su centro de control de misión ubicado en las instalaciones de la NASA.

2. Requerimientos del programa

Comandos:

- **Comandos de desplazamiento:** que le permiten moverse sobre la superficie
    - Tipo_movimiento: Puede ser avanzar o girar.
    - Magnitud es el valor o cantidad del movimiento
    - Unidad_medida es la unidad con la que se hace la medición del movimiento.
- **Comandos de análisis:**
    - Tipo_analisis puede ser fotografiar, composición o perforar
    - Objeto es el nombre del elemento sobre el cual se hace el análisis
    - Comentario es un valor opcional que permite agregar información sobre el análisis a realizar o el elemento que se analizará, este comentario debe ingresarse entre comillas simples.

Los tres comandos deben estar separados cada uno por el carácter *pipe* **‘|’**, todos los comandos inician con un carácter “0” o “1” que sirve para reconocer de qué tipo es comando, “0” comandos de desplazamiento y “1” comandos de análisis.

Identificación de Elementos:

- **Comandos de puntos de interés**, que incluyen las ubicaciones geográficas de componentes o elementos hallados en el terreno
    - Tipo_elemento puede ser roca, crater, monticulo o duna.
    - Tamaño es el valor de la dimensión del elemento
    - Unidad_medida es la unidad con la que se realizó la medición del tamaño del elemento
    - Coordenada_x es la posición sobre el eje x en el plano cartesiano del elemento
    - Coordenada_y es la posición sobre el eje y en el plano cartesiano del elemento

3. Diseño TAD

a. Curiosity

- **Datos mínimos:**

  list<sComando> : Lista de tipo sComando que almacena todos los comandos que se encuentren en el archivo txt y los comandos ingresados por el programa.

  list<sElemento> : Lista de tipo sElemento que almacena todos los elementos que se encuentren en el archivo txt y los comandos ingresados por el programa.

  float orientacion : Número que permite el almacenamiento de la orientación del curiosity.

  float coords : Array de tipo float que permite conocer la posición del Curiosity.

- **Operaciones:**

  getComandos() : Retorna el comando deseado de la lista de comandos.

  getElementos() : Retorna el elemento deseado de la lista de elementos.

  cargarComandos(namefile) : Carga en memoria los comandos de desplazamiento contenidos en el archivo identificado por el nombre del archivo que recibe, es decir, utiliza adecuadamente las estructuras lineales para cargar la información de los comandos en memoria.

  agregarComando(comando) : Agrega el comando que recibe de movimiento descrito a la lista de comandos del robot Curiosity. El movimiento puede ser de dos tipos: avanzar o girar.

  cargarElementos(namefile) : Carga en memoria los datos de puntos de interés o elementos contenidos en el archivo identificado por el nombre del archivo que recibe, es decir, utiliza adecuadamente las estructuras lineales para cargar la información de los elementos en memoria.

  agregarElementos(sElemento elemento): Agrega a la lista de tipo Elemento la información consignada.

  actualizarOrientacion(nuevaorientacion) : Actualiza la orientación del Curiosity, recibe

  simularComandos() : Ejecuta todos los comandos presentes en la lista de comandos que tiene el Curiosity.

  setComandos(comandos) : Fija en la lista de comandos que pertenecen al robot, recibe una lista de comandos.

  setElementos(elementos) : Fija en la lista de elementos que pertenecen al robot, recibe una lista de elementos.

  setOrientacion(orientación) : Fija la orientación del robot, recibe una orientación.

  setCoords(cordX, cordY) : Fija las coordenadas del robot, recibe dos coordenadas.

  guardar(namefile, typefile) : Guarda en el archivo la información solicitada de acuerdo al tipo de archivo, comandos de movimiento y de análisis que debe ejecutar el robot, recibe el nombre de un archivo y el tipo de archivo que es.

  insertarElementos() : Función para insertar elementos en una lista.

ubicarElementos() : Función para ubicar elementos en el espacio.
Limites(list<sElemento> elementos) : Función para determinar los límites del espacio basados en una lista de elementos.
Quadtree* crearQuadtree() : Función para crear un Quadtree.
insertarElementosQuadtree(Quadtree*quadtree,list<sElemento>elementos): Función para insertar elementos en un Quadtree.
EnCuadrante(float X1, float Y1,float X1, float Y2) : Función para buscar un elemento en un Quadtree.

b. Comandos

- **Datos mínimos :**

sMovimiento: Conformado por:

- **TipoMovimiento :** Puede ser avanzar o girar .
- **Magnitud :** Del movimiento a realizar.
- **UnidadMedida :** Medición en metros del movimiento.

Almacena los datos de los movimientos que contiene el archivo txt y los que fueron agregados por el programa.

sAnálisis: Conformado por:

- **TipoAnalisis:** El proceso a ejecutar por el robot.
- **Objeto:** Que elemento va a utilizar para el analizar.
- **Comentario:** Un comentario que salga del análisis realizado.

Almacena los datos de los análisis que contiene el archivo txt y los que fueron agregados por el programa.

c. Elementos

- **Datos mínimos :**

tipoElemento: Almacena cual es la clasificación del elemento.
tamaño: Almacena el tamaño del elemento.
unidadMedida: Medición en metros del elemento.
coorDx: Coordenada que mide la posición en X.
coorDy: Coordenada que mide la posición en Y.

d. Punto

- **Datos mínimos :**

Int cord_X: Número que permite el almacenamiento de la coordenada **X** del punto.

Int cord_Y: Número que permite el almacenamiento de la coordenada **Y** del punto.

e. QuadTree

- **Datos mínimos:**

Bool vacío: Bandera que almacena el valor de verdad del estado de un quadtree.

Bool hoja: Bandera que almacena el valor de verdad del estado de una hoja.

Punto punto: Objeto clase punto que permite almacenar un punto en el quadtree.

Int min_X : Número que permite determinar el límite inferior en el eje **X** del quadtree.

Int max_X : Número que permite determinar el límite superior en el eje **X** del quadtree

Int min_Y : Número que permite determinar el límite inferior en el eje **Y** del quadtree.

Int max_Y : Número que permite determinar el límite superior en el eje **Y** del quadtree

Ul : Puntero que conecta con la parte superior Izquierda.

Dl : Puntero que conecta con la parte inferior izquierda.

Ur : Puntero que conecta con la parte superior derecha.

Dr : Puntero que conecta con la parte inferior derecha.

- **Operaciones:**

Quadtree() : Constructor del quadtree vacío.

Bool colisiona(const Punto& punto1, const Punto& punto2) : Comprueba si dos puntos colisionan en el Quadtree.

dividir() : Divide el Quadtree en subcuadrantes.

insertar(const Punto& p) : inserta un nuevo punto en el Quadtree.

buscarEnQuadtree(int x, int y) : Busca un punto en el Quadtree basado en las coordenadas (x, y).

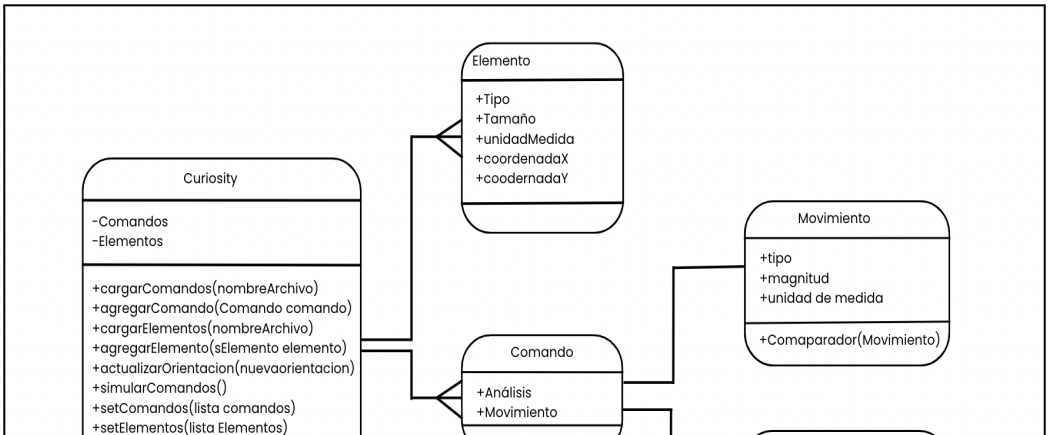
preorden (Quadtree *arbol) : Realiza un recorrido preorden en el Quadtree.

contarPuntos() : Cuenta los puntos que existen.

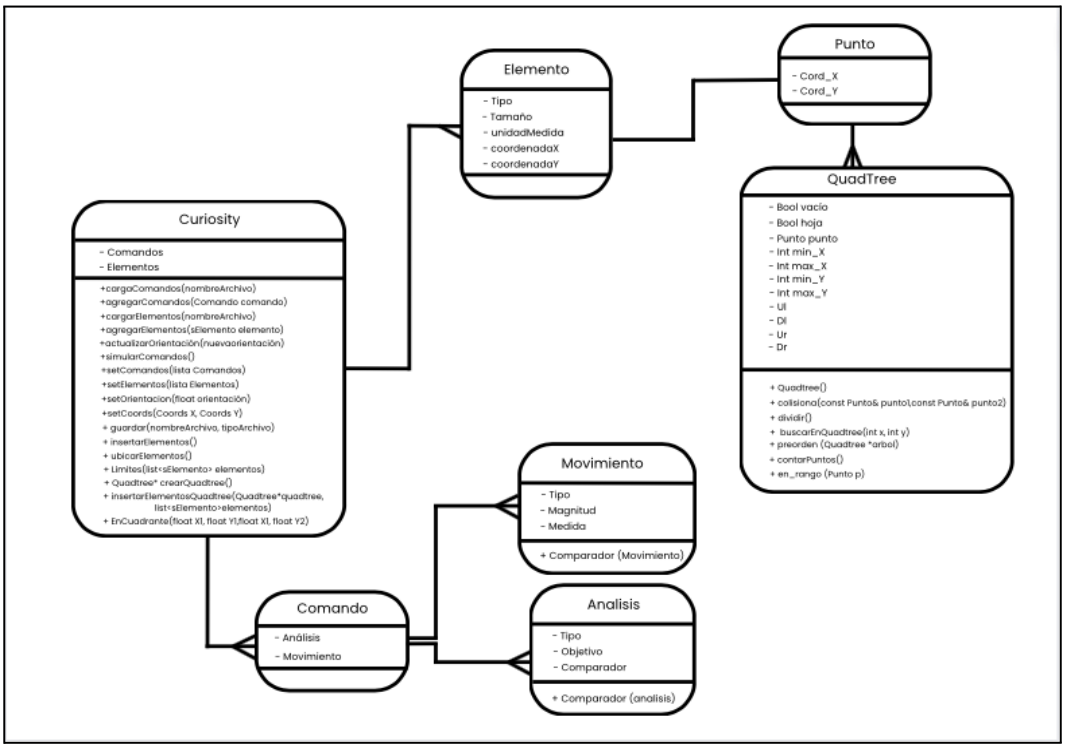
en_rango (Punto p): Verificar que el punto se encuentre en el rango del Quadtree.

4. Diagrama de Relación

- **Entrega 1:**



- **Entrega 2:**



5. Plan de Pruebas

Función:

Curiosity.simularComandos() es un método que itera sobre una lista de comandos y ejecuta las acciones correspondientes. A continuación se presentarán los pasos formulados para realizar un plan de pruebas de la función:

1. Verificar el funcionamiento correcto de la función para diferentes tipos de comandos. Para ello, se hizo uso de un archivo llamado "comandos.txt" en el que se tenía información acerca de lo los avances o giros que debía

2. Verificar que las coordenadas y la orientación del robot se actualicen correctamente después de cada movimiento.

.

3. Verificar que el método no tiene efectos secundarios no deseados, como cambiar el estado interno de la clase Curiosity de forma incorrecta.

Comandos:

```
$cargar_comandos comandos
Comandos cargados:
avanzar,10,metros
fotografiar,roca,'roca grande xd'
avanzar,10,metros
girar,45,grados
avanzar,30,metros
fotografiar,roca,'roca grande xd'
girar,15,grados
perforar,roca,'roca aun mas grande xd'
avanzar,10,metros
```

PLAN DE PRUEBAS: METODO CARGAR_COMANDOS			
DESCRIPCIÓN	**DATOS ENTRADA**	**VALORES ESPERADOS**	**VALORES OBTENIDOS**
1) Coordenadas Iguales en X y Y	X: 1 Y Y: 1	11 - 1 fotografiar - roca - roca grande xd 21 - 1 45 42.21 - 22.21 fotografiar - roca - roca grande xd 60 perforar - roca - roca aun mas grande xd 47.21 - 30.87	11 - 1 21 - 1 45 42.21 - 22.21 60 47.21 - 30.87
2) Coordenadas diferentes en X y Y	X: 2 Y: 5	12 - 5 fotografiar - roca - roca grande xd 22 - 5 45 43.21 - 26.21 fotografiar - roca - roca grande xd 60 perforar - roca - roca aun mas grande xd 48.21 - 34.87	2 - 5 22 - 5 45 43.21 - 26.21 60 48.21 - 34.87
3) Coordenadas		10 - 5 fotografiar - roca - roca grande xd 20 - 5 45	10 - 5 20 - 5 45 41.21 - 26.21 60

diferentes en X y Y	X: 0 Y: 5	41.21 - 26.21 fotografiar - roca - roca grande xd 60 perforar - roca - roca aun mas grande xd 46.21 - 34.87	46.21 - 34.87

Los valores esperados junto con los valores obtenidos son coincidentes en X y Y, la única falla es la ausencia en comandos de análisis pues no ejecuta ninguno, la posible solución es la correcta validación del análisis en la parte de simular los comandos.

Evidencia Fotográfica:

```
$simular_comandos 2 5
Coordenadas actuales: 12 5
Coordenadas actuales: 22 5
Orientacion actual: 45
Coordenadas actuales: 43.21 26.21
Orientacion actual: 60
Coordenadas actuales: 48.21 34.87
```

```
$simular_comandos 1 1
Coordenadas actuales: 11 1
Coordenadas actuales: 21 1
Orientacion actual: 45
Coordenadas actuales: 42.21 22.21
Orientacion actual: 60
Coordenadas actuales: 47.21 30.87
```

```
$simular_comandos 0 5
Coordenadas actuales: 10 5
Coordenadas actuales: 20 5
Orientacion actual: 45
Coordenadas actuales: 41.21 26.21
Orientacion actual: 60
Coordenadas actuales: 46.21 34.87
```

Función:

Curiosity.en_cudrante() es un método recursivo, en el cual se le pasan por parámetros los límites máximos y mínimos de un "Cuadrante" del cual se quiere conocer los elementos que allí yacen:

1. Se crea una lista de tipo 'sElemento' llamada "elementos_en_cuadrante", aquí se guardaran los elementos encontrados y por consiguiente la lista que se retornara.
2. Se crea una auxiliar de tipo 'Punto' "p" la cual será una variable auxiliar para poder encontrar las coordenadas guardadas en memoria del árbol.
3. Se empezar a iterar sobre la lista elementos igualando las coordenadas del elemento con las de la variable auxiliar p (por cada una de las iteraciones).
4. Posteriormente ingresa en un condicional, donde se llama la función de la clase Quadtree denominada "buscar" que devuelve un valor booleano si encuentra el elemento, posteriormente imprime el elemento que encontró y lo guarda en la lista.

ELEMENTOS:

```
1  roca|0.25|metros|45|20
2  crater|0.3|metros|100|20
3  crater|2|metros|1|20
4  duna|0.5|metros|5|30
5  roca|0.7|metros|65|73
6  monticulo|1|metros|23|35
7  crater|4|metros|84|-23
8  duna|1|metros|47|39
9  crater|4|metros|-29|80
```

PLAN DE PRUEBAS: METODO EN_CUADRANTE			
DESCRIPCIÓN	**DATOS ENTRADA**	**VALORES ESPERADOS**	**VALORES OBTENIDOS**
1) Coordenadas Iguales en X	X_1: 0 X_2: 100 Y_1: 0 Y_2: 100	roca 0.25 metros cráter 0.3 metros cráter 2 metros duna 0.5 metros roca 0.7 metros montículo 1 metros duna 1 metros	roca 0.25 metros cráter 0.3 metros cráter 2 metros duna 0.5 metros roca 0.7 metros montículo 1 metros duna 1 metros
2)Coordenadas diferentes en X y Y	X_1: 0 X_2: 20 Y_1: 0 Y_2: 40	crater 2 metros duna 0.5 metros	crater 2 metros duna 0.5 metros
3)Coordenadas diferentes en X y Y	X_1: 0 X_2: 85 Y_1: -40 Y_2: 0	crater 4 metros	crater 4 metros

Los valores esperados junto con los valores obtenidos son coincidentes en cuanto a que se muestran los mismos elementos, se observa que para todas las combinaciones hechas se obtienen los resultados que se esperan, los cuadrantes ingresados logran mostrar los detalles de las coordenadas pertenecientes a cada objeto, funcionamiento correcto.

Evidencia Fotográfica: