Práctica 6. Mallas regulares

Sesiones de prácticas: 2

Objetivos

Utilizar mallas regulares para realizar búsquedas eficientes por rangos.

Descripción de la EEDD

En la nueva versión del diseño que aparece más abajo, se añaden los puntos de recarga donde hay que dejar la moto en caso de que se quede sin batería. Los puntos de recarga son lugares fijos en el mapa, tienen una ubicación y un id único en el sistema. Se cargan en una malla regular con la siguiente funcionalidad:

```
template <class T>
class MallaRegular {
        …
public:
   MallaRegular(float aXMin, float aYMin, float aXMax, float aYMax, int
nDivX, int nDivY);
   T buscarCercano(float x, float y);
   bool fueraAmbito (float x, float y);
   unsigned maxElementosPorCelda();
   unsigned mediaElementosPorCelda();
}
```

Podemos asumir que todo tipo T con el que se instancie la clase tiene los métodos getX() y getY() implementados. El constructor ahora contempla un número diferente de divisiones para las filas y para las columnas.

La función *buscarCercano(float x, float y)* localiza el punto fijo (puntos de recarga) de una localización dada por sus coordenadas (x,y) (moto). La función *fueraAmbito(float x, float y)* indica si la ubicación dada por las coordenadas (x,y) (moto) está (true) o no (false) fuera del ámbito de un punto fijo (punto recarga), es decir, que no haya ningún punto fijo a menos de una casilla de distancia.

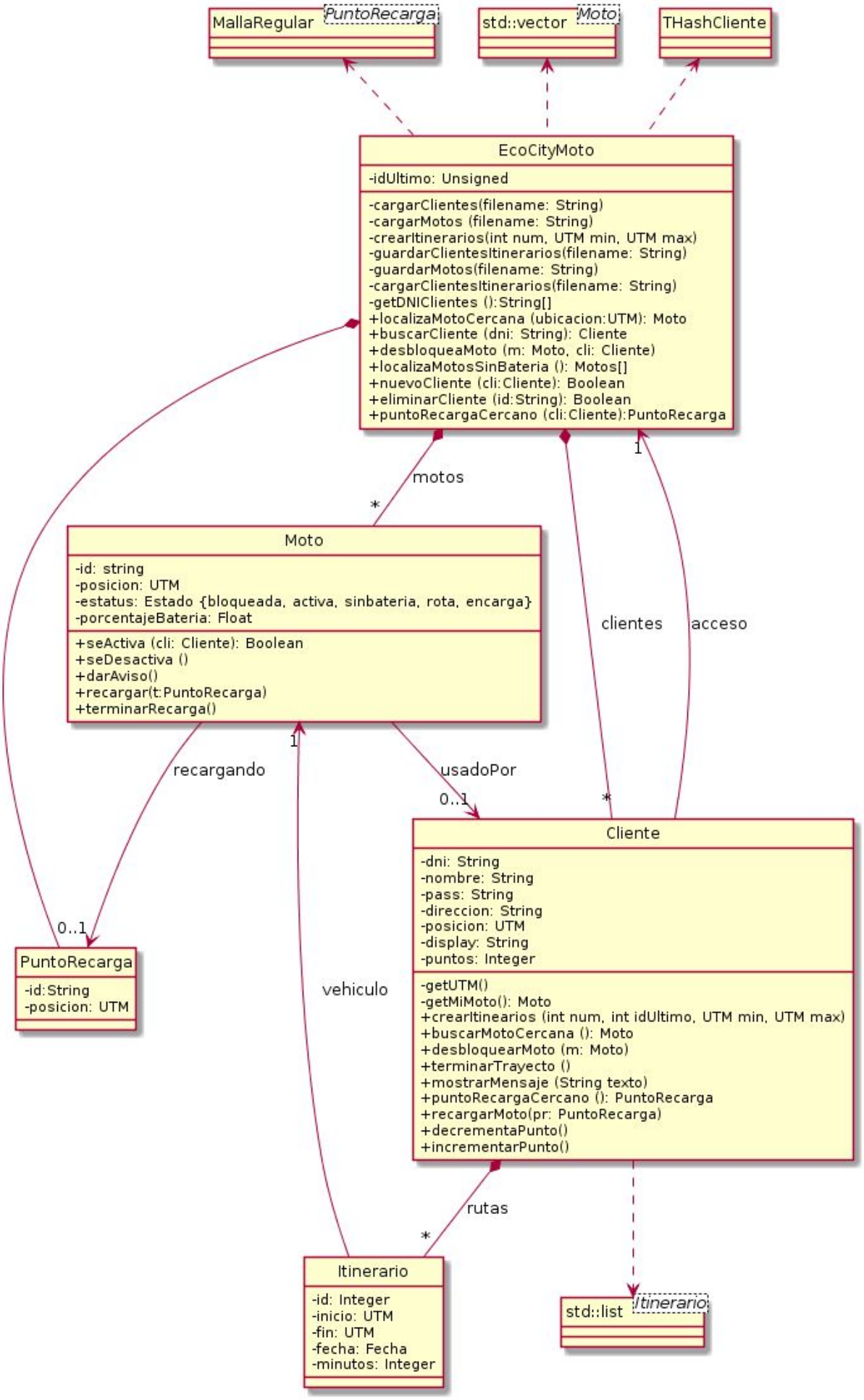
Diseño del modelo:

En esta última versión del diseño UML vamos a introducir la clase *PuntoRecarga*. Éste debe ser el lugar donde se debe dejar la moto obligatoriamente si queda en estado sinBateria. Si no fuera así, el cliente se ve amonestado con un punto menos en su puntuación inicial de 10 con Cliente::decrementarPunto(). Si el cliente llega a 0 se le da de baja en el sistema. Si por el contrario lo deja en su lugar, se le añade un punto positivo.

En este nueva versión de la aplicación, cuando el cliente se queda sin batería (<15%) y lleva la moto a un punto de recarga, se llama a *Cliente::puntoRecargaCercano()* que le devuelve el punto donde debe ir llamando a EcoCityMoto::puntoRecargaCercano(Cliente:cli) que la busca de forma eficiente en la malla regular.

Cuando se le da el punto de recarga, debe conducir a este punto y llamar a *Cliente::recargarMoto(pr: PuntoRecarga)*. Esta función debe conectar la moto con su punto de recarga en *Moto::recargar(pr: PuntoRecarga)*, y cambiar convenientemente los estados de la moto al 100% de batería. Cuando el proceso termina, se llama a *Moto::terminarRecarga()* que la desconecta del terminal de carga. Comprobad que el cliente ha cambiado su posición a una idéntica al punto de recarga. Si hace esto bien, es recompensado con un punto positivo.

El usuario debe estar informado en todo momento a través de su terminal con *mostrarMensaje(String: texto)*. El resto de la funcionalidad del sistema no cambia.


MallaRegular
PuntoRecarga
std::vector
Moto
THashCliente
EcoCityMoto
-idUltimo: Unsigned
-cargarClientes(filename: String)
-cargarMotos (filename: String)
-crearItinerarios(int num, UTM min, UTM max)
-guardarClientesItinerarios(filename: String)
-guardarMotos(filename: String)
-cargarClientesItinerarios(filename: String)
-getDNIClientes ():String[]
+localizaMotoCercana (ubicacion:UTM): Moto
+buscarCliente (dni: String): Cliente
+desbloqueaMoto (m: Moto, cli: Cliente)
+localizaMotosSinBateria (): Motos[]
+nuevoCliente (cli:Cliente): Boolean
+eliminarCliente (id:String): Boolean
+puntoRecargaCercano (cli:Cliente):PuntoRecarga
motos
*
1
Moto
-id: string
-posicion: UTM
-estatus: Estado {bloqueada, activa, sinbateria, rota, encarga}
-porcentajeBateria: Float
+seActiva (cli: Cliente): Boolean
+seDesactiva ()
+darAviso()
+recargar(t:PuntoRecarga)
+terminarRecarga()
clientes
acceso
recargando
usadoPor
1
0..1
*
0..1
PuntoRecarga
-id:String
-posicion: UTM
vehiculo
Cliente
-dni: String
-nombre: String
-pass: String
-direccion: String
-posicion: UTM
-display: String
-puntos: Integer
-getUTM()
-getMiMoto(): Moto
+crearItinearios (int num, int idUltimo, UTM min, UTM max)
+buscarMotoCercana (): Moto
+desbloquearMoto (m: Moto)
+terminarTrayecto ()
+mostrarMensaje (String texto)
+puntoRecargaCercano (): PuntoRecarga
+recargarMoto(pr: PuntoRecarga)
+decrementaPunto()
+incrementarPunto()
rutas
*
Itinerario
-id: Integer
-inicio: UTM
-fin: UTM
-fecha: Fecha
-minutos: Integer
std::list
Itinerario

Programa de prueba:

1. Crear de forma aleatoria pero siempre las mismas posiciones para unos 300 puntos de recarga en el rango de Jaén, (latitud, longitud): (37, 3) - (38, 4).
2. Probar dos posibles tamaños de malla y quedaos con aquella más grande que tenga como máximo 5 puntos de recarga en una casilla. Mostrar el tamaño final de la malla por pantalla.
3. Añadir a todos los clientes un número de puntos aleatorio entre 1 y 10.
4. Añadir un nuevo cliente que no exista previamente con coordenadas en Jaén.
5. Localizar el cliente anterior dado su DNI y mostrar toda la información del cliente.
6. Buscar la moto más cercana al cliente anterior (que se pueda utilizar) y mostrar la información de la moto.
7. Modificar ahora la función EcoCityMoto::localizaMotosSinBateria() para que encuentre las motos sin batería que están fuera del radio de acción de un punto de recarga. Se considera que está fuera en caso de que esté más alejado que el ancho de una casilla de la malla regular.
8. Realizar un itinerario (dentro de la provincia de Jaén) con la moto localizada. La posición inicial viene dada por la posición de la moto, y la final será aleatoria. Hay que tener en cuenta que si la batería está por debajo del 15% hay que recargar la moto. Por tanto, para ver la funcionalidad que se ha añadido en esta práctica, hay que forzar a que el recorrido sea lo suficientemente largo como para recargar la moto.
    a) Buscar el punto de recarga más cercano, mostrarlo por pantalla
    b) Recargar la moto. Mostrar el estado de la batería antes y después de recargarla.
    c) Actualizar el estado de la moto en todo momento e informar al cliente mediante el display.
    d) Incrementar el número de puntos del cliente, mostrarlo por pantalla. Si tiene 10 puntos, hay que indicar que no se ha podido incrementar ya que dispone del número máximo posible.

Estilo y requerimientos del código:

1. El código debe ser claro, tener un estilo definido y estar perfectamente indentado, para ello se pueden seguir algunos de los estilos preestablecidos para el lenguaje C++ (http://geosoft.no/development/cppstyle.html).

2. Deben comprobarse todas los posibles errores y situaciones de riesgo que puedan ocurrir (desbordamientos de memoria, parámetros con valores no válidos, etc.) y lanzar las excepciones correspondientes, siempre que tenga sentido. Leer el tutorial de excepciones disponible en el repositorio de la asignatura en docencia virtual.

3. Se valorará positivamente la calidad general del código: claridad, estilo, ausencia de redundancias, etc.