Índex

1-Introducció

Avui en dia el modelatge de ciutats és un problema obert pel que no existeixen solucions estàndards ni de codi lliure. Aquest és un problema força important per a indústries com la del cinema, la realitat virtual o els videojocs. Dins d'aquest problema es poden presentar sub-problemes interessants, dels quals el modelatge de les xarxes de carrers pren un rol fonamental. La complexitat geomètrica que té una ciutat fa que el modelatge d'aquest tipus d'estructures sigui una tasca gens menyspreable.

Qualsevol persona, amb més o menys relació amb el món dels ordinadors, haurà jugat a algun joc d'aventures gràfiques on l'escenari escollit és una gran ciutat, o haurà vist alguna pel·lícula relativament actual on el protagonista viu la seva aventura envoltat de grans edificis. Cada vegada més es fan servir eines informàtiques per a crear aquests escenaris monumentals. Molts d'aquests escenaris han estat creats manualment, és a dir, muntant els edificis un per un fins aconseguir una ciutat.

El modelatge procedural pretén solucionar bona part d'aquest problema utilitzant les seves funcionalitats per a generar de forma sistemàtica la geometria necessària en cada cas. A grans trets, el procediment consisteix en deixar a la màquina la feina de crear la ciutat pròpiament dita i així l'usuari només s'ha de preocupar d'introduir els paràmetres necessaris per aconseguir el seu propòsit.

Existeixen eines de desenvolupament procedural, però el seu ús es troba restringit a unes quantes aplicacions que, generalment, no inclouen xarxes de carrers.

1.1 Objectius

L'objectiu d'aquest projecte és el desenvolupament d'una eina de generació de xarxes de carrers a partir d'exemples. L'eina ha de permetre generar una xarxa de carrers nova que sigui semblant a l'existent en un mapa vectorial donat.

A més, també es preten unir aquesta aplicació amb l'urbanEngine per tal de poder generar vistes en 3D sobre aquestes xarxes de carrers, a més d'ampliar les opcions de l'urbanEngine a l'hora de crear ciutats.

1.2 Metodologia

La tècnica que s'ha seguit per a la creació d'aquesta aplicació es la que es troba explicada a l'article "Interactive Example-Based Urban Layout Synthesis" (D. Aliaga, C. Vanegas i B. Benes, 2008) que va ser presentat durant el SIGGRAPH ASIA del 2008. En ell presenten un sistema interactiu per a la sinterització de ciutats mitjançant exemples, que consisteix en:

- **Entrada**: Exemples de fragments de ciutats(mapa Vectorial)
- **Processament**: L'usuari marca les pautes perquè l'aplicació generi una nova xarxa de carrers.

- **Sortida**: Nova xarxa de Carrers.

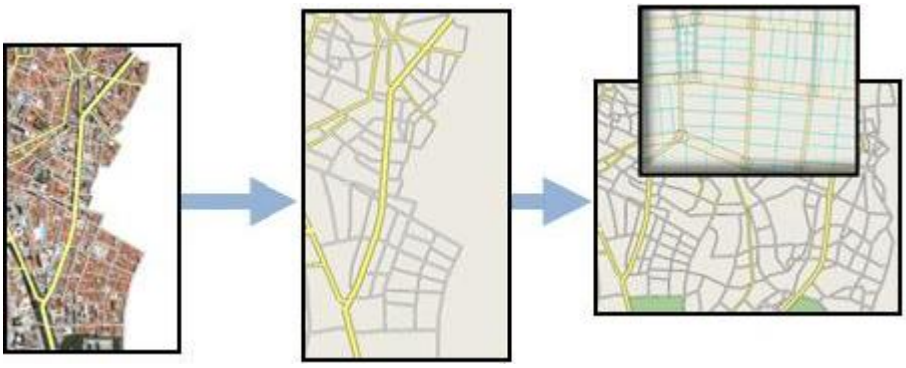


Figura 1: Diagrama on es mostra el procés que s'ageix l'aplicació.

(per a una descripció molt més detallada sobre la tècnica seguida, veure l'apartat 3).

Pel desenvolupament d'aquest projecte s'ha utilitzat com a metodologia l'UML (Unified Modeling Language), que tot i no ser una metodologia, és un llenguatge de modelat estàndard pel camp de l'enginyeria del programari. Es pot entendre com a tal.

L'UML s'utilitza per definir un sistema, per detallar els elements del sistema, per documentar i construir. Per aconseguir això l'UML disposa de varis tipus de diagrames que mostren diversos aspectes dels elements representats.

En aquest projecte s'han utilitzat:

- Els diagrames de casos d'ús (veure apartat 4.1.1.1)
- Els diagrames d'activitat (veure apartat 4.2)
- Els diagrames de classe (veure apartat 4.3)
- Els diagrames de seqüència (veure apartat 4.4)

1.3 Temporització

Aquest projecte es va iniciar a principis de febrer de 2009 i s'ha acabat a finals d'agost de 2009.

Els primers dos mesos es van dedicar a l'aprenentatge del llenguatge de programació Python i a la recerca d'un format de mapes vectorials lliure per poder-lo utilitzar com a format d'entrada dels exemples. A finals de Març es va trobar el format de mapes OSM que complia tots els requeriments, les següents dos setmanes es van dedicar a entendre i saver utilitzar aquest format. A partir d'aquest moment es va començar a escriure el codi següint la mateixa seqüencia que seguria l'aplicació:

- Carregar Mapa
- Calcular Estadistiques
- Generar nova xarxa de carrers

A principis de juliol, quan es va tenir tot el codi principal enllestit, es va començar amb el proces de la creació de la UI i de lligar tot el codi realitzat amb l'urbanEngine. Finalment, durant l'últim mes es va acabar de redactar la memòria. La Figura 2 mostra un diagrama de Gantt amb la temporalització del projecte.

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Fase d'aprenentatge	2/3/2009	4/3/2009	8.8s
2	Python	2/3/2009	3/23/2009	7s
3	Recerca Format	2/3/2009	3/23/2009	7s
4	Format OSM	3/23/2009	4/3/2009	2s
5	Implementació	4/3/2009	7/31/2009	17.2s
6	Carregar Mapa	4/3/2009	4/16/2009	2s
7	Calcular Estadístiques	4/16/2009	5/18/2009	4.6s
8	Generar Xarxa de Carrers	5/18/2009	6/24/2009	5.6s
9	UI	6/24/2009	7/9/2009	2.4s
10	Unió amb l'urbanEngine	7/9/2009	7/31/2009	3.4s
11	Documentació	4/3/2009	9/1/2009	21.6s

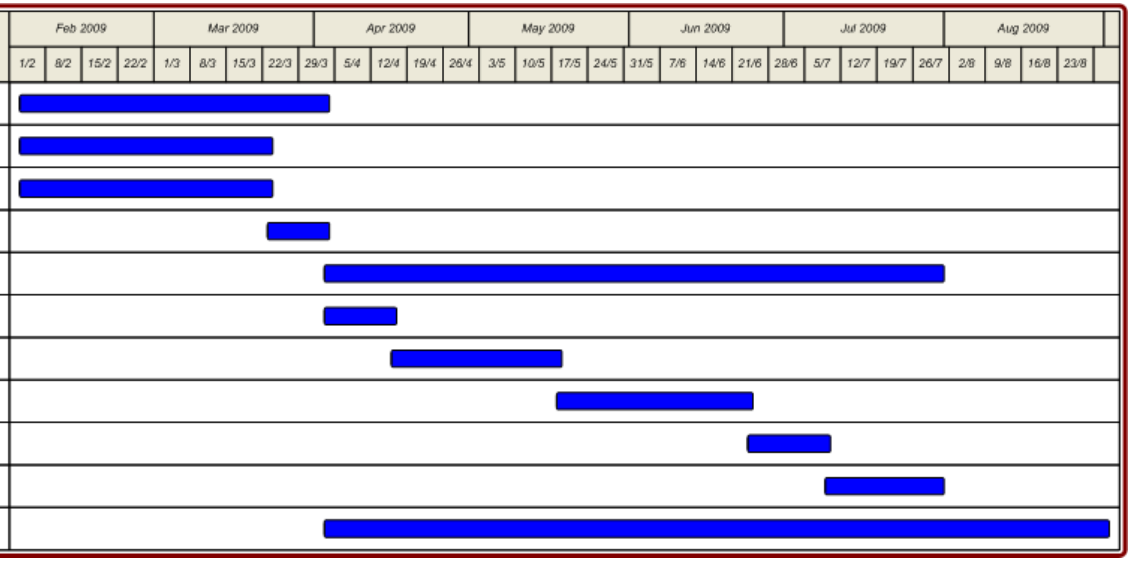


Figura 2: Diagrama de Gantt.

1.4 Estructura del document

Aquesta memòria està estructurada en cinc capítols fonamentals que recullen tota la documentació i explicació necessària del projecte.

- El primer capítol conté la introducció al tema central del projecte així com els objectius que es van marcar al inici del mateix i la temporalització estructurada.

- En el segon capítol s'expliquen detalladament les eines, llibreries i formats escollits: Houdini, Python, OSM, UrbanEngine, miniDOM, wxPython i wxGlade.

- El tercer capítol mostra en detall la tècnica seguida per construir aquesta aplicació.

- El quart capítol consta de l'anàlisi dels requeriments del sistema amb els diagrames de casos d'ús

- El cinquè capítol consta de l'anàlisi del sistema, diagrames d'activitat, diagrama de classes i diagrames de seqüència.

- El sisè capítol és complementari al cinquè, ja que s'hi mostra la implementació d'algun del mètodes presentats en aquest i la implementació de la UI.

- El setè capítol recull els resultats obtinguts amb exemples de ciutats construïdes amb el programa.

- El vuitè capítol mostra les conclusions extretes dels resultats obtinguts.

- Finalment, el capítol 9 proposa el treball futur que es pot desenvolupar a partir del projecte actual.

2- Programari Utilitzat

Per a la realització d'aquest projecte bàsicament s'han utilitzat tres eines diferents:

- El Houdini com a modelador 3D per tal de poder mostrar els carrers i les ciutats 3d que és genera l'urbanEngine.

- El Python com a llenguatge de programació base. Juntament amb ell s'han utilitzat les llibreries miniDom, per llegir i generar arxius en format OpenStreetMap, i les WxPython, per construir la interfície d'usuari.

- El format de mapes OpenStreetMap com a format d'entrada del exemples.

En un principi és va plantejar utilitzar els mapes de "Google maps", però el fet de necessitar una llicencia per poder descarregar els mapes i la previsió d'evitar possibles problemes futurs, van fer que es preferís buscar una opció gratuïta i lliure com és el format OpenStreetMap, que serà explicat més detalladament a la secció 2.3.

2.1 Houdini

El Houdini Master és una eina per a la creació avançada d'afectes Visuals 3D, que permet controlar l'animació des de la mateixa interfície o utilitzant un dels dos llenguatges d'script que incorpora. El primer d'ells, i més antic, és l'anomenat HSript. Aquest llenguatge és exclusiu de Houdini i existent des de les primeres versions. El segon és el Python, introduït a partir de la versió 9.0 amb l'objectiu d'estandarditzar el funcionament i dotar el programa d'un codi conegut.

Houdini
3D ANIMATION TOOLS

El Houdini treballa usant un sistema de nodes en forma de graf acíclic per a representar els objectes de l'escena. D'aquesta manera s'obté un sistema d'objectes independents, units entre ells per un flux de dades.

La Figura 3 mostra l'entorn de treball del Houdini. Aquest entorn està compost per una barra d'eines (superior), des d'on es generen totes les figures i accions. La zona de treball és l'espai on es poden visualitzar els resultats usant diferents vistes i es pot seleccionar i modificar objectes. L'arbre de nodes és l'espai on es representen tots els objectes de l'escena en forma de caixetes independents enllaçades les unes a les altres. Finalment, un cop seleccionat un node, podem consultar i modificar les seves propietats a través del sistema de pestanyes que facilita la interfície.

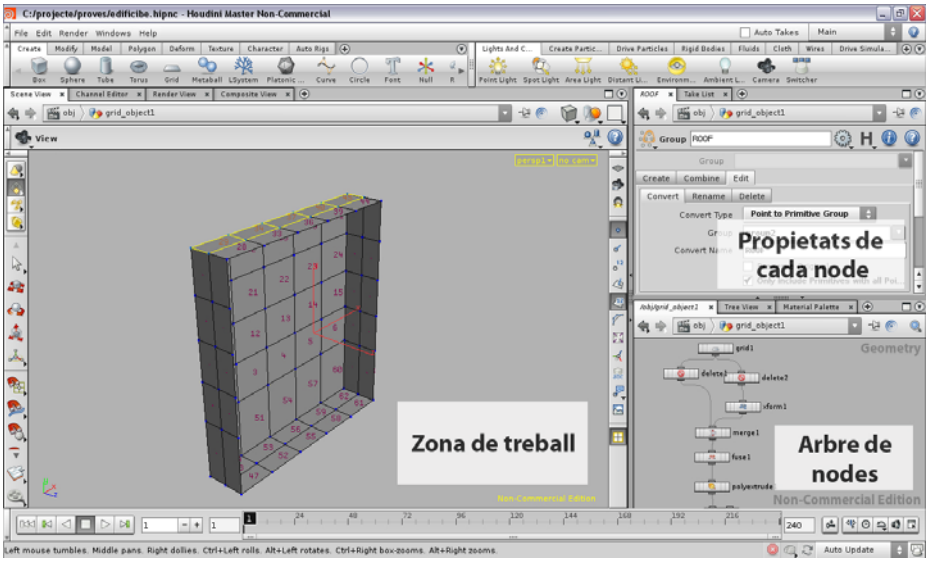


Figura 3: Captura de l'entorn de treball del Houdini.

El Houdini divideix les xarxes de nodes en diferents categories:

- VEX Builder (VOP) – Xarxes per al llenguatge del Houdini anomenat VEX.
- Objects (OBJ) – Xarxes d'objectes (geometria, llum, càmera, etc).
- Geometry (SOP) – Xarxes per construir geometries.
- Particle (POP) – Xarxes per efectes de partícules.
- Compositing (COP2) – Xarxes per composició 2D.
- Dynamic (DOP) – Xarxes per a efectes dinàmics.
- Channel (CHOP) – Xarxes per a operacions de canals. Els canals controlen com canvien els valors a través del temps.
- Shaders (SHOP) – Xarxes per a crear diferents shaders.
- Output (ROP) – Xarxes per especificar les opcions de renderització.

El tipus que s'utilitza en aquest projecte són els SOP.

2.2 UrbanEngine

L'urbanEngine és una eina de modelatge procedural de ciutats i xarxes de carrers que permet modelar ciutats reals a partir d'una imatge, però també permet generar ciutats sintètiques a partir de patrons.

Les dades provinents d'aquesta aplicació es tracten d'una manera bastant semblant al cas de treballar amb imatges reals, primerament es dibuixen les illes al houdini. Un cop s'ha fet això es criden els mètodes de l'urbanEngine per a dividir les illes en parcel·les i finalment aixecar els edificis. A continuació es mostra la seqüencia que es segueix per dibuixar el resultat d'aquesta aplicacio:

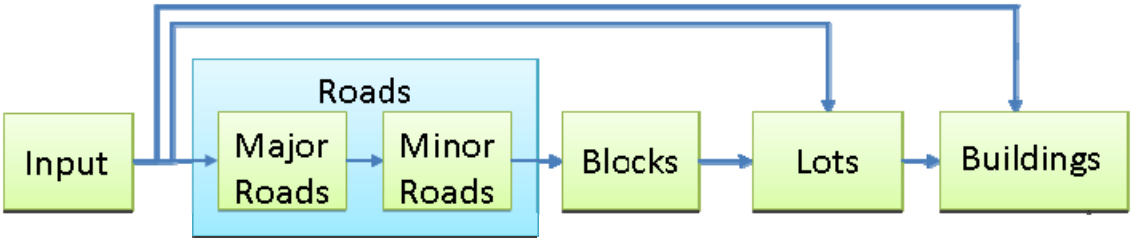


Figura 4: Diagrama on es mostra el procés que s'ageix l'urbanEngine a l'hora de dibuixar una ciutat 3D.

A l'hora de construir les geometries, l'urbanEngine les llegeix d'un fitxer XML que pot ser editat per l'usuari per generar ciutats més d'acord amb els seus gustos. En el cas d'aquesta aplicació, s'anomena uE_EBStreetGeneratorConfig.xml.

2.3 OpenStreetMap

OpenStreetMap es un projecte col·laboratiu l'objectiu del qual es crear y facilitar dades geogràfiques lliures a qualsevol persona que els pugui necessitar.

El projecte es va iniciar degut a que en moltes parts del mon, los dades geogràfiques (geodades) publiques no son d'us lliure. Normalment en aquest països s'ha delegat la tasca de d'obtenir aquest tipus d'informació a diverses entitats independents del govern (l'IGN en el cas d'Espanya), les quals venen aquesta cartografia a qui la necessiti.

En països como ara els Estats Units, los dades cartogràfiques en brut (sense tractar) que pertanyen al govern són de domini públic, però les versions editades i corregides solen tenir drets d'autor.

El problema d'utilitzar aquestes geodades amb drets d'autor és que hi haurà algunes coses que no sempre ens permetran fer, com ara canviar el nom d'un carrer o utilitzar-les en un programa. A més per controlar l'ús que se'n fa, solen introduir-hi dades errònies, anomenades ous de pascua, en forma d'elements inexistents, topònims imaginaris, marques d'aigua digitals o punts de control minúsculs. Aquest control de fet només serveix per poder verificar als jutjats si algú ha fet servir les dades sense el pagament corresponent.

2.3.1 Creació de Mapes

Els mapes es creen seguint aquest 5 passos:

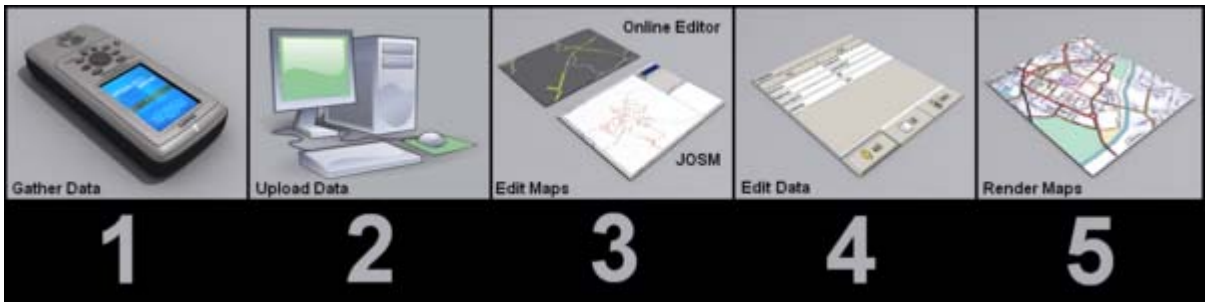


Figura 5: Diagrama on es mostra el procés que es sagueix per a la creació d'uns mapa OSM

1- **Recollir les dades**

   Hi ha diverses formes de recollir dades per a OSM:

   - GPS - És manera més usual de recollir les dades per a OSM i és essencial per les àrees ben mapejades.

- Imatges de Yahoo!, Landsat y mapes NPE (Regne Unit)
- Fotografies o mapes – Assegurat que siguin completament lliures per a copiar i utilitzar en OSM.

2- Pujar les dades

Els passos següents són necessaris per pujar les dades a OSM, i descarregar-les a JOSM, l'editor de OpenStreetMap més utilitzat, per poder editar-les.

- Guardar els arxius en GPX

  Si has recollit les dades amb un GPS, es necessita guardar-les en format GPX, alguns GPS permeten fer-ho directament. En els que no, s'ha de guardar en algun altre format, per exemple en .tcx, i convertir-lo en GPX utilitzant un programa com ara el GPSBabel.

- Pujar a OSM

  Les dades es poden pujar a OSM abans o després de convertir-les en mapes. Totes les dades GPX que s'utilitzen per a la creació de mapes es recomana que es pugin per a ús públic de manera que totes les dades OSM tinguin un origen i que d'altres les pugin utilitzar.

  Per poder pujar dades a OSM s'han de seguir els següents passos:

  1- Anar a la pagina principal de OSM.

  2- Iniciar sessió.

  3- Clickar "GPS traces" a la part superior.

  4- Clickar a "See just your traces, or upload a trace".

  5- En les tres capces de la part superior fer el següent:

     1. Buscar i seleccionar l'arxiu que es vol pujar.

     2. Anomenar l'arxiu.

     3. Afegir etiquetes útils per poder trobar-ho fàcilment.

  6- Si es vol que els mapes siguin accessibles per els altres (recomanat) clickar l'opció Públic.

  7- Clickar a "Upload".

  8- Les dades tardaran una mica a ser incorporades a dins de OSM, un cop finalitzat aquest temps es podrà accedir immediatament a elles.

  9- Un cop fets tots els passos anteriors, ja es poden descarregar les dades al JOSM juntament amb la resta de dades de l'àrea.

- Descarregar les dades al JOSM

  Per descarregar les dades al JOSM des de la pagina del OSM S'han de seguir les següents instruccions.

  1- Clickar "Download".

  2- Per seleccionar l'àrea que es desitja descarregat s'ha de fer:

     1. Obrir un arxiu GPX i descarregar l'àrea que es veu per pantalla.

     2. Obtenir les dades des de la pagina del OSM.

        a. Buscar l'àrea que es vol editar.

        b. Clickar a la barra "View", o a Permalink en el costat inferior esquerra.

        c. Copiar i enganxar el link a la capsa descarrega del JOSM.

  3- Escollir el que es vol descarregar:

     1. Dades originals: Dades obtingudes directament dels GPSs.

     2. Dades del mapa: Mapa en format OSM per ser editada.

  4- Clickar "Download".

  5- Repetir per múltiples àrees si és necessari.

3- Generar/Editar les dades OSM

Les dades OSM es generen a partir dels següents elements:

- Nodes: Punts definits per la seva latitud i longitud. Poden en utilitzar com a principi i final dels segments o sols.
- Vies: Línees que uneixen dos o mes nodes.
- Àrees: Son les línees que formen un bucle complert. S'utilitzen per cartografiar àrees.

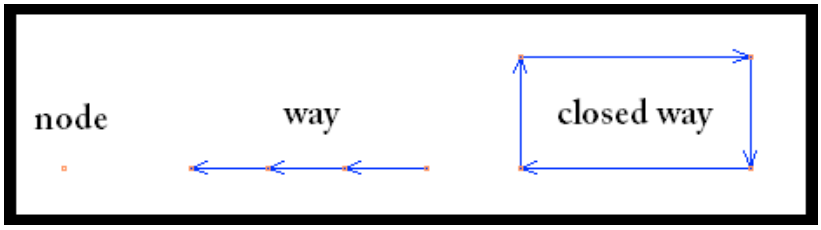


Figura 6: Diagrama on es mostren els diferents elements que s'utilitzen per generar/editar dades OSM.

4- Etiquetar les dades i afegir detalls

Un cop es té una via completa, es necessita afegir etiquetes (Tags) per a poder veure-les renderitzades en el mapa. Les etiquetes són parells de valor-clau. Existeixen moltes etiquetes, algunes més comunes que altres. Quan es fa servir JOSM, etiquetar s'aconsegueix afegint parells de valor-clau al panell de Propietats/Membres.

5- Renderitzar els mapes

Un cop s'han pujat les dades creades/modificades en els punts anteriors, per poder veure'n els canvis realitzats es pot escollir entre dues opcions.

- Generar mapes de bits renderitzats des de les dades que s'han modificat utilitzant una eina de renderitzat des del PC. El projecte OpenStreetMap en té tres de diferent:

    1. **Kosmos** : el Kosmos es una eina lleugera de renderització de mapes OSM, dissenyada per poder ser utilitzada pels usuaris en el seu ordinador. Segurament la més senzilla de les tres opcions.

    2. **Osmarender**: Un renderitzador basat en "Extensible Stylesheet Transformation (XSLT)" que és capaç de crear "Scalable Vector Graphics (SVG)", que pot ser visionat directament amb algun navegador web o convertit a mapa de bits.

    3. **Mapnik**: Un renderitzador molt ràpid escrit en C++ que genera mapes de bits.

- Fer que les dades pareixin al visualitzador de mapes de la pagina principal de OpenStreetMap. La pagina principal conté un '+' al costat dret. Després de clickar el '+' normalment s'accedeix a les opcions de la 'Capa Base' 'Mapnik' i 'Osmarender'. Totes dues representen el mapa generat per el renderitzadors explicats anteriorment.

    1. **Mapnik**: La 'Capa Base' del Mapnik s'actualitza aproximadament cada hora. Les illes s'actualitzen automàticament quan les augmentes, tot i que han de tenir una antiguitat de 3 o més. Per forçar la seva actualització s'ha d'afegir "/dirty" al final de l'adreça.

    2. **Osmarender**: La 'Capa Base' de l'Osmarender s'actualitza automàticament quan es pugen dades noves. Aquest procés pot durar un cert temps, després del qual ja s'haurien de veure les modificacions que s'han fet. S'haurà de fer un refresc de la pagina per poder-los veure.

2.3.2 Característiques dels mapes

OpenStreetMap no té cap tipus de restricció en el que fa referència al contingut de "claus" i "valors" que s'assignen a Nodes, Vies o Àrees. No obstant això, per facilitar

les coses, hi ha un conjunt de característiques (juntament amb les corresponents claus i valors) definides, perquè els softwares genèrics les pugin interpretar i mostrar.
El contingut bé donat per les idees recollides a la pagina "Labels" de l'OSM (per veure'n tots els valors possibles accedir a l'annex1). No es pretén que aquesta sigui una llista completa; es poden afegir propostes per a noves característiques a la pagina "proposed features" per a la seva discussió i consideració.

Només es permet una clau per element(Node, Via o Àrea).

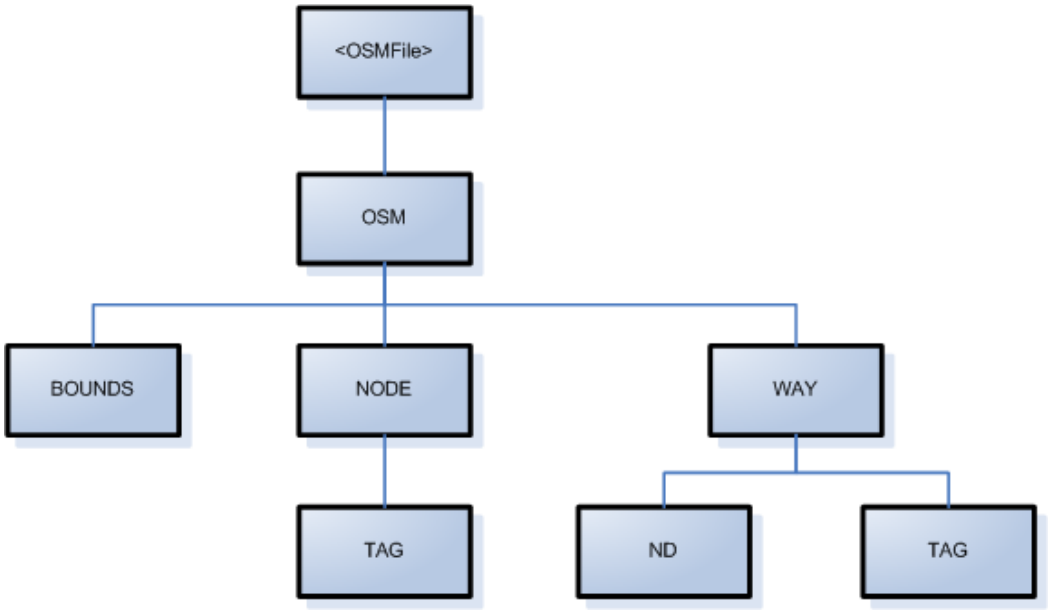


Figura 7: Exemple de l'estructura d'un arbre OSM amb el miniDOM

2.3.3 Format OSM

Els fitxers OSM són fitxers XML amb un format especial per poder representar dades OpenStreetMap. Les dades primitives que els formen són els nodes, les vies i les relacions que juntament amb la capçalera seran explicats a continuació.

2.3.3.1 Capçalera

Tots els fitxers .osm comencen amb una capçalera on s'especifiquen una sèrie de valors:

- **xml version**: versió de l'xml que s'ha utilitzat.
- **encoding**: tipus de codificació utilitzada.
- **osm version**: versió del protocol osm utilitzat.
- **generator**: sempre és la cadena OpenStreetMap server
- **minlat**: latitud mínima del mapa.
- **minlon**: longitud mínima del mapa.
- **maxlat**: latitud màxima del mapa.

- **maxlon**: longitud màxima del mapa.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap server">
  <bounds minlat="41.96126" minlon="2.82495" maxlat="41.96736"
maxlon="2.83283"/>
```

Figura 8: Exemple d'una Capçalera OSM

2.3.3.2 Node

Un node és un parell latitud/longitud. S'utilitza per construir blocs per altres elements, com per exemple els carrers, o per si mateixos, com ara en el cas de punts d'interès.

- **id**: Es tracta d'un valor enter igual o més gran que 1. Les ids no són úniques ja que per exemple un carrer pot tenir la mateixa id que un node.
- **user**: L'autor de l'última modificació
- **timestamp**: La data i l'hora de l'última modificació.
- **visible**: Indica si l'objecte ha estat eliminat o no es troba a la base de dades
- **tags**: Es tracta d'un parell clau/valor, en que una clau no es pot repetir dos cops dins el mateix node. Per a veure'ls accedir a l'annex1.
- **lat**: Coordenada latitud. Pot ser un nombre entre el -90 i el 90. Es guarda en format de 7 decimals.
- **lon**: Coordenada longitud. Pot ser un nombre entre el -180 i el 180. Es guarda en format de 7 decimals.

```
<node id="31022931" lat="41.9642059" lon="2.8317523" version="4"
 changeset="128272" user="Xevi" uid="100124" visible="true"
 timestamp="2009-04-03T19:18:19Z">
    <tag k="highway" v="bus_stop"/>
</node>
```

Figura 9: Exemple d'una node OSM amb una parada d'autobusos assignada a ell.

2.3.3.3 Vies

Una via és una llista de almenys dos nodes que descriuen un element lineal com pot ser un carrer o similar. Els nodes poden ser membres de més d'un carrer.

- **id**: Igual que en el Node
- **user:** Igual que en el Node
- **timestamp**: Igual que en el Node

- **tags**: Igual que en el Node
- **nodes**: Es tracta d'un llistat amb tots les ids dels nodes que formen el carrer.

```
<way id="4826122" visible="true" timestamp="2009-03-2T00:50:24Z"
 version="4" changeset="794288" user="Xevi" uid="100124">
    <nd ref="31022926"/>
    <nd ref="31022927"/>
    <nd ref="359254131"/>
    <nd ref="31022928"/>
    <nd ref="31022929"/>
    <nd ref="31022930"/>
    <nd ref="359254110"/>
    <nd ref="31022931"/>
    <nd ref="31022932"/>
    <tag k="name" v="Carrer Universitat de Girona"/>
    <tag k="created_by" v="Potlatch 0.10f"/>
    <tag k="highway" v="residential"/>
    <tag k="layer" v="2"/>
</way>
```

Figura 10: Exemple d'un carrer OSM

2.3.3.4 Relacions

Una relació és un grup de zero o més dades primitives amb rols associats. S'utilitza per especificar relacions entre objectes i també pot modelar un objecte abstracte.

- Id: Igual que en el Node.
- User: Igual que en el Node.
- Timestamp: Igual que en el Node.
- Tags: Igual que en el Node
- Members: Una llista de primitives amb un rol associat.

```
<relation id="3" visible="true" timestamp="2008-10-16T16:36:07Z"
 version="1" changeset="374143" user="Matt" uid="70">
    <member type="way" ref="4253333" role="street"/>
    <member type="way" ref="27776917" role="house"/>
    <member type="way" ref="27776918" role="house"/>
    <member type="way" ref="27776919" role="house"/>
    <tag k="type" v="associatedStreet"/>
  </relation>
```

Figura 11: Exemple d'una relació OSM

2.4 Python

Python és un llenguatge de programació creat per Guido van Rossum en l'any 1990. Es compara habitualment amb TCL, Perl, Scheme, Java i Ruby. En l'actualitat Python es desenvolupa com un projecte de codi obert, administrat per la Python Programari Foundation. L'última versió estable del llenguatge és actualment la 3.1 (27 de Juny de 2009) .

Python és considerat com la "oposició lleial" a Perl, llenguatge amb el qual manté una rivalitat amistosa. Els usuaris de Python consideren a aquest molt més net i elegant per a programar. Python permet dividir el programa en mòduls reutilitzables des d'uns altres programes Python. També hi ha mòduls inclosos que proporcionen E/S de fitxers, crides al sistema, sockets i fins a interfícies a GUI (interfície gràfica amb l'usuari) GTK, Qt, WxWidgets(WxPython) entre altres...

Python és un llenguatge interpretat, el que estalvia un temps considerable en el desenvolupament del programa, perquè no és necessari compilar ni enllaçar. L'intèrpret es pot utilitzar de manera interactiva, el que facilita experimentar amb característiques del llenguatge, escriure programes d'un sol ús o provar funcions durant el desenvolupament del programa.
El principal objectiu que persegueix aquest llenguatge és la facilitat, tant de lectura, com de disseny.

2.4.1Python dins de Houdini

Actualment encara no està acabada tota la implementació de Python dins de Houdini. Això fa que la majoria de vegades sigui impossible trobar documentació sobre les funcions i procediments que disposa el programa.

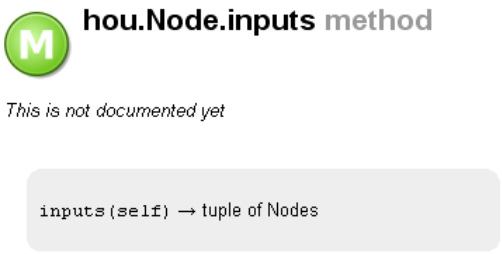


Figura 12: Missatge que es mostra quan la documentació que es cerca encara no està feta.

2.4.1.1 Utilitzar Python

Primer de tot cal aclarir les dues maneres de fer servir Houdini. La primera, i més habitual, és utilitzant la interfície d'usuari del programa. La segona és fent servir la consola de Python. Usant la consola es poden fer totes les accions disponibles a la interfície gràfica.

Com que Python és un llenguatge interpretat, es pot executar funcions en temps real, tal i com es faria prement un botó del programa.

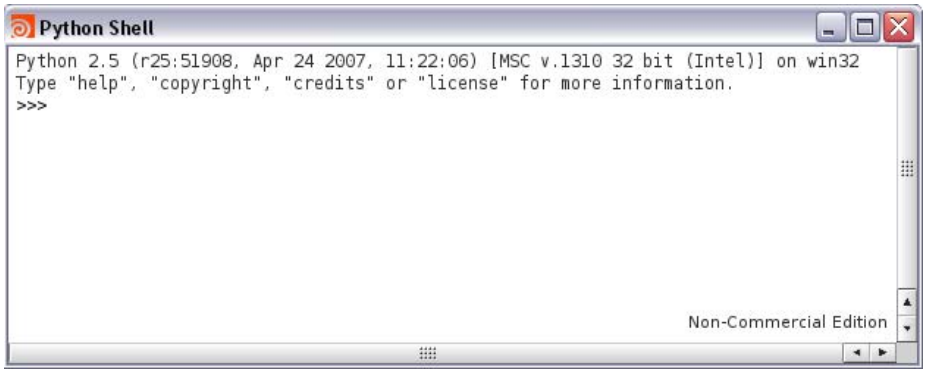


Figura 13: Captura de la consola de Python. Per accedir-hi: Windows > Python Shell.

El sistema Python que té Houdini és exactament el mateix que es pot descarregar des de la web oficial de Python. L'única diferència és la incorporació d'un mòdul addicional anomenat ***hou*** que conté totes les funcionalitats, objectes i propietats de Houdini. Aquesta API rep el nom de HOM (*Houdini Object Model*).

Un exemple creant un BOX amb la consola de Python. Primer de tot es s'importa el mòdul *hou*. Seguidament es crea un objecte de tipus '*geo*' que emmagatzemarà tots els objectes. Finalment es crea el box utilitzant al sentència "*createNode()"*.

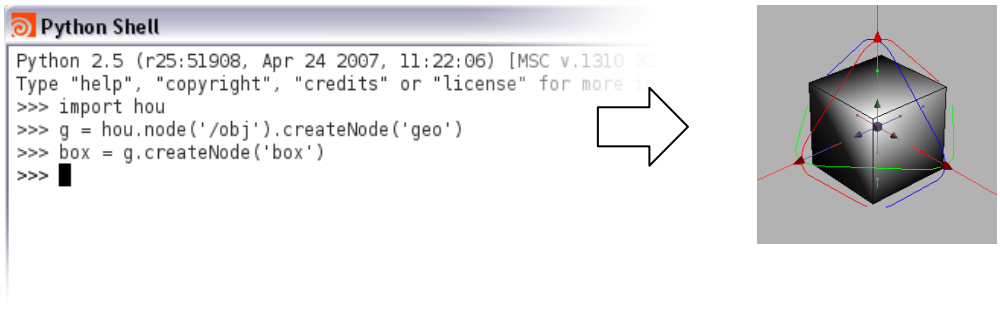


Figura 14: Creació d'un cub utilitzant el Python.

També es poden aplicar transformacions a través de la consola. En aquest cas es crea un node POLYEXTRUDE i s'enllaça amb el BOX utilitzant la sentència "*setFirstInput()*". Llavors es canvia el paràmetre *tz* del node POLYEXTRUDE.

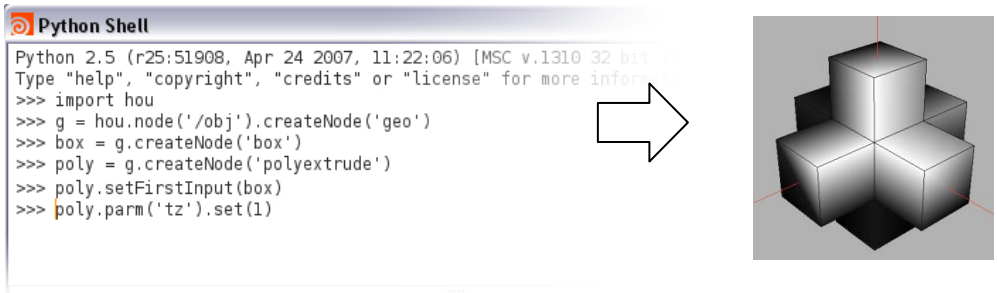


Figura 15: També es poden aplicar transformacions fent servir la consola.

També és possible carregar scripts des de fitxers externs. Aquesta és la manera que s'ha triat per a executar el projecte.

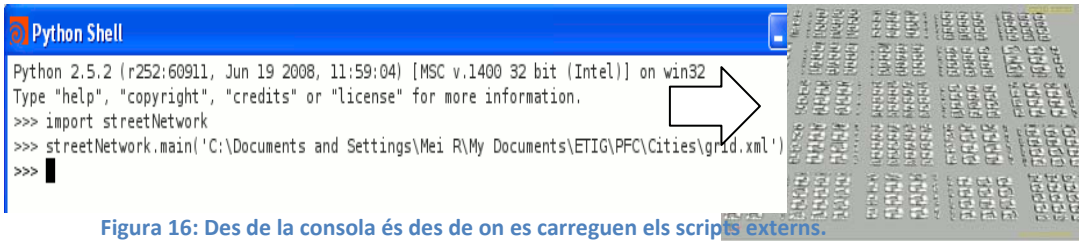


Figura 16: Des de la consola és des de on es carreguen els scripts externs.

Un cop s'ha importat un script aquest queda desat en memòria. Si es fa algun canvi al fitxer de l'script, només cal executar la sentència *reload(mòdul)* i s'actualitzarà la càrrega.
Cada vegada que es crea un objecte dins de Houdini aquest rep un nom que apunta a un espai de memòria on hi ha desades les dades de l'objecte. Quan aquesta creació es fa des de la interfície gràfica, aquest nom és invisible a l'usuari, que només pot veure l'etiqueta que rep l'objecte. En canvi, quan l'objecte és creat des de Python, la manera d'accedir a aquest és mitjançant el nom. En aquest cas, l'etiqueta es comporta com una propietat més, accessible a través dels mètodes apropiats. Aquesta etiqueta pot ser modificada, però amb la restricció que ha de ser única.

2.4.2 miniDOM

El miniDOM és una implementació lleugera del Document Object Model interface (DOM), l'objectiu de la qual és definir un parser XML que simplifiques la feina de parsejar amb SAX. Per a més informació sobre el DOM es pot accedir a http://www.w3.org/DOM/.

El SAX per la seva banda va ser creat per solucionar alguns dels problemes del DOM, per exemple el fet que en DOM s'ha de crear un objecte que contingui tot el document, si el document és molt llarg això no resulta pràctic. En canvi el SAX crea tota una estructura en arbre n-ari. El problema és que el SAX només proporciona informació en brut pel que, per obtenir l' informació estructural, s'han d'utilitzar patrons de disseny per fer-ho possible. Tot i així resulta una feina bastant pesada pel programador.

Un objecte miniDOM utilitza un parser SAX amb tasques preassignades a etiquetes especifiques. Quan es processa una etiqueta es genera un objecte que representa el node d'interès.

Un cop un document ha seguit parcejat amb miniDOM s'obté una estructura en forma d'arbre formada per nodes miniDOM.

2.4.2.1 nodes miniDOM

El nodes miniDOM poden ser de diferents tipus, a continuació es mostraran tots els tipus possibles però només es descriuran els més comuns:

- Document: Es tracta del node arrel de l'arbre que conté el document.
- DocumentFragment: Es tracta d'un node molt semblant a l'anterior però molt més lleuger, normalment es tracta d'un fragment d'un document.

- DocumentType: Node que s'utilitza com a tribut d'un node del tipus Document, s'utilitza per especificar el format del fitxer original.
- EntityReference: Es tracta d'un node que guarda una EntityReference que es trobava en el document.
- Element: Es tracta del node base que ens permet accedir a tota la resta
- Attr: Es un node que guarda un parell clau/valor.
- ProcessingInstruction: Representa una instrucció de processament.
- Comment: Aquest tipus de node guarden els comentaris que es troben en el fitxer original.
- Text: Es tracta d'un node que guarda un string al seu interior
- CDATASectionEntity: Aquest nodes s'utilitzen per marcar zones en la que s'ignoren les funcions assignades a caràcters especials.
- Notation: Representa una notació

A continuació es llistaran i explicaran sèrie de propietats i mètodes dels nodes Element.

- Accés als nodes
    - **childNodes**:Retorna una llista amb totes les referències als Nodes fills.
    - **firstChild**: Retorna una referència al primer fill.
    - **lastChild**: Retorna una referència a l'últim fill
    - **nextSibling**: Retorna una referència al següent fill del pare del node.
    - **previousSibling**: Retorna una referència a l'anterior fill del pare del node.
- Treballar amb els Atributs dels nodes
    - **nodeTyp**e: Retorna 1 si és un node tipus tag, un 2 si és tipus atrib i un 3 si es tipus text.
    - **nodeName**: Retorna el nom del node.
    - **parentNode**: Retorna una referència al node pare.
    - **getElementById**: Retorna una referència al node amb la id especificada.

- **getElementsByTagName**: Retorna una llista amb una referència a cadascun dels nodes amb el tag desitjat.
- **specified**: Retorna un booleà indicant si el node te l'atribut especificat.
- **hasAttributes**: Retorna un booleà indicant si el node té atributs definits.
- **getAttribute**: Retorna el valor de l'atribut especificat.
- **setAttribute**: Assigna el valor especificat al l'atribut que es desitgi.
- **removeAttribute**: S'elimina l'atribut especificat.
- **data**: Retorna el valor del node en els nodes tipus text i *undefined* a la resta.
- **nodeValue**: Retorna el valor del node en els nodes tipus text i *null* a la resta.
- **attributes**: Retorna una llista amb els atributs del node.

- Afegir/eliminar nodes
  - **createElement**: Crea un element del tipus especificat
  - **createTextNode**: Crea un node de tipus text.
  - **appendChild**: Afegeix un fill nou al final de la llista.
  - **insertBefore**: Afegeix un fill nou al davant del seleccionat.
  - **replaceChild**:Substitueix un fill pel node especificat.
  - **removeChild**: Elimina un fill.
  - **cloneNode**: Copia el node.

2.4.3 WxPython

Les wxPython són un conjunt de llibreries per a la creació de GUIs utilitzant el llenguatge de programació Python. Aquestes permeten als programadors de Python crear programes amb una Interfície d'usuari robusta, simple i fàcilment. Estan implementades com a mòduls d'extensió del Python que embolcallen les conegudes wxWindows unes llibreries de GUIs multiplataforma que estan escrites en C++.

Es van escollir aquestes llibreries en comptes de les QtPhython (són les dues úniques opcions per poder crear interfícies d'usuari en Python dins de Houdini), ja que les wxPython són totalment lliures i no estan lligades a cap empresa, en canvi les QtPhython actualment pertanyen a Nokia. De moment, es distribueixen sota una llicencia GPL per la qual són lliures d'ús i de modificació, però això podria canviar cosa que podria acabar portant problemes de llicencies.

Com el Python i les wxWindows, les wxPython són de codi lliure, el que significa que el pot fer servir qualsevol i que el codi font és accessible per llegir-lo o modificar-lo.

Les wxPython són multiplataforma. Això vol dir que amb una mica de compte es pot executar el mateix codi en diversos sistemes operatius sense cap modificació. Les plataformes suportades actualment són Microsoft Windows, la majoria de sistemes Unix o els basats en aquest que disposen de les llibreries GTK i l'OS X 10.3.9 o superior.

2.4.3.1 Utilitzar WxPhython amb Houdini

Per poder utilitzar les WxPhython amb el Houdini primer s'han de seguir aquest passos.

1. Esbrinar quina versió del Python és troba integrada en la versió del Houdini que s'utilitza. Per poder-ho saber s'ha d'anar al menú Windows→Python Shell, un cop s'ha obert la consola, la versió es troba a la part superior esquerra.

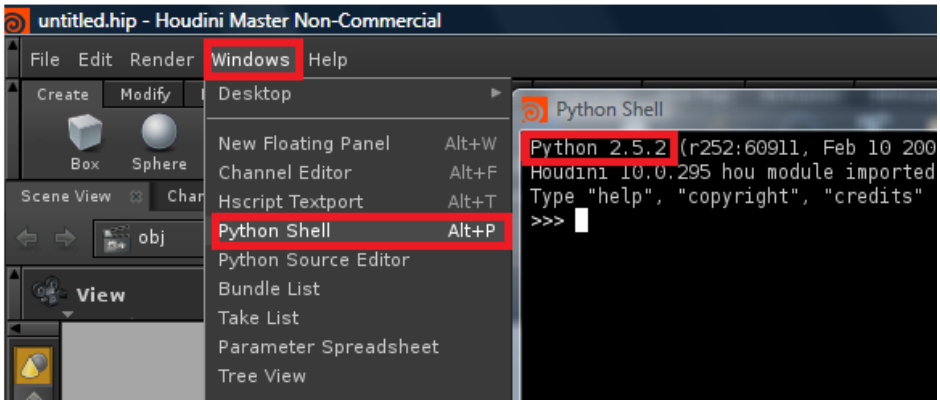


Figura 17: Imatge os es mostren els passos per accedir a la consola de Python que porta el Houdini.

2. Descarregar i instal·lar la mateixa versió que té el houdini des de [http://www.python.org]

3. Descarregar i instal·lar les wxPython que corresponen a la versió del python de [http://wxpython.org/download.php].
   Només la versió ansi està suportada actualment pel Houdini.

4. Copiar la carpeta wx de la ubicació on s'han instal·lat les wxPython. S'instal·len dins de la carpeta del Python que hem instal·lat anteriorment. Per exemple "C:\Python25\Lib\site-packages\wx-2.8-msw-ansi"

5. Enganxar la carpeta dins de la carpeta d'instal·lació del Houdini en la subcarpeta houdini\scripts\python. Per exemple "C:\Program Files (x86)\Side Effects Software\Houdini 10.0.295\houdini\scripts\python"

2.4.3.2 wxGlade

wxGlade és un programa per dissenyar GUIs escrit amb Python utilitzant les llibreries wxPython. Amb l'wxGlade es poden crear Interfícies d'usuari de wxWidgets/wxPython. En aquests moments pot generar codi Python, C++, Perl, Lisp i XRC.

wxGlade no és (ni ho serà) una IDE complerta, sinó que només és un dissenyador; el codi generat no fa res més apart de mostrar la finestra creada.

Quan l'obres es mostren 3 finestres:

- La de l'arbre, on es mostra l'estructura i objectes del projecte.
- La de propietats, on es mostren les propietats de l'objecte seleccionat.
- La del menú principal, on es poden seleccionar les diferents opcions i deixar-les anar en la nostra GUI.
- Una vegada s'ha seleccionar al menú principal el tipus de finestra que volem crear (Diàleg o Frame), és mostra una quarta finestra on es mostra el Diàleg o Frame que acabem de crear.

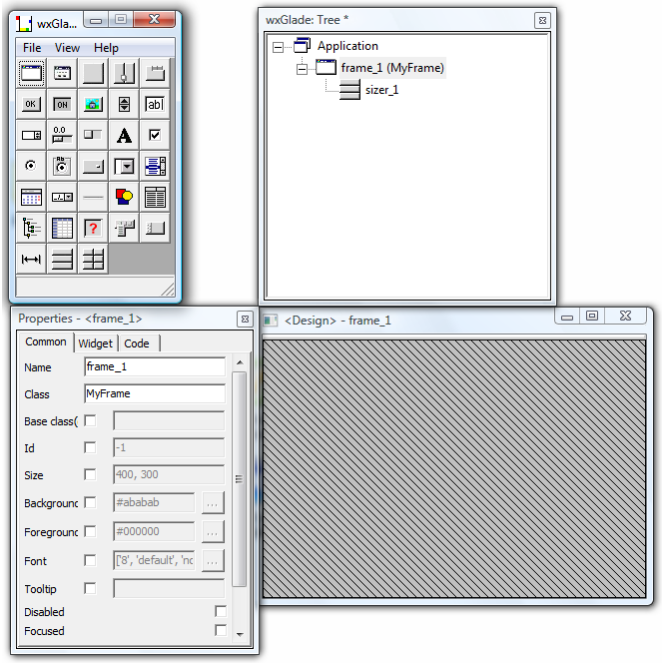


Figura 18: Imatge os es mostren els menus de l'wxGlade.

3- Modelatge de xarxes de carrers a partir d'exemples

En aquest capítol es procedirà a fer una presentació del modelatge de xarxes de carrers a partir d'exemples, així com s'explicaran algunes de les solucions que s'han escollit per tal de resoldre alguna de les tasques proposades per aquest projecte. Veure figura 19.

- **Entrada**: Exemples de fragments de ciutats(mapa Vectorial)
- **Processament**: L'usuari marca les pautes perquè l'aplicació generi una nova xarxa de carrers.
- **Sortida**: Nova xarxa de Carrers.

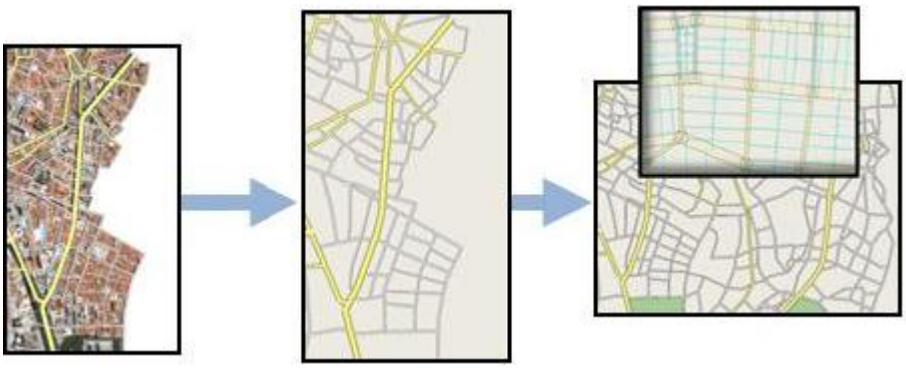


Figura 19 Diagrama on es mostra el procés que s'ageix l'aplicació.

3.1 Processament de l'entrada

L'objectiu d'aquest processament és obtenir una sèrie d'atributs que descriguin estadísticament la geometria dels carrers i les illes, per tal que després es pugui generar una nova geometria amb el mateix estil. En aquesta aplicació només es permeten mapes OSM com a entrada. Veure figura 20.

Per més informació sobre aquest tipus de mapes mirar l'apartat 2.3

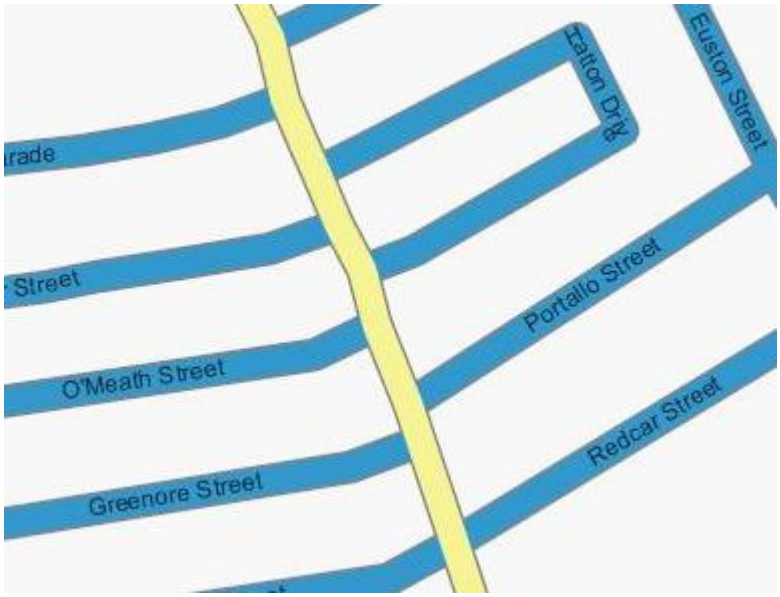


Figura 20: Exemple d'un mapa OSM

D'un mapa OSM en podrem extreure:

- La geometria dels carrers (línia central del carrers)
- La geometria de les illes (contorn de les illes)

S'utilitza la informació del mapa de vectors per trobar les interseccions, veure figura 21.

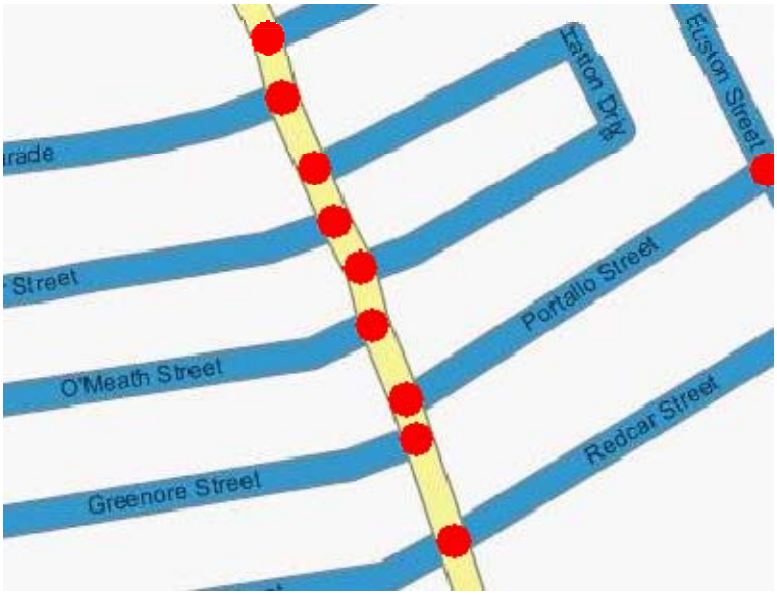
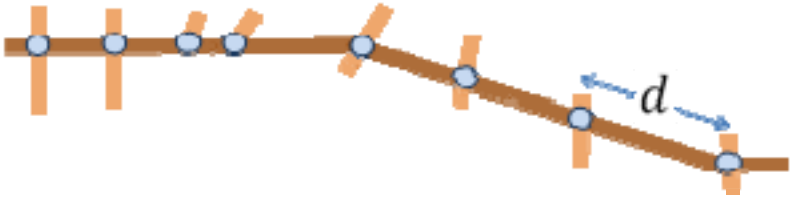


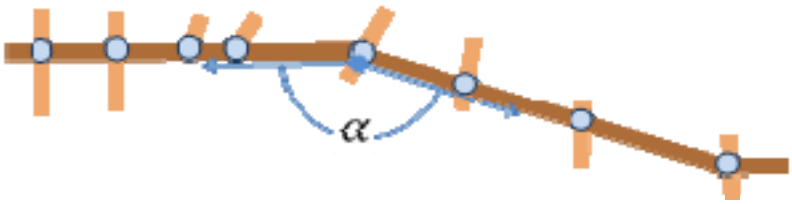
Figura 21: Exemple d'un mapa OSM amb les interssecions Marcades

Un cop s'ha fet això, el següent és calcula,de cadascun dels carrers, les següents propietats:

- La mitjana de la distància *d* entre dues interseccions consecutives del mateix carrer.



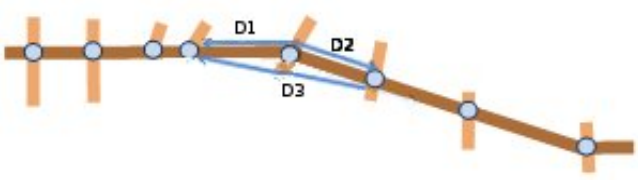
- La mitjana de l'angle *α* entre dos segments (tant dels segments intersecció com dels que no) consecutius del mateix carrer.



- La variança de la distància *d* entre els segments que passen per cadascuna de les interseccions respecte al promitg de les mitjanes de la distància de tots els carrers que passen per ella.

- La variança de l'angle *α* entre dos segments (tant dels segments intersecció com dels que no) consecutius del mateix carrer respecte a la mitjana de l'angle del carrer.

- També es calcula un valor al que hem anomenat "tortuositat", que representa la mitjana de la relació entre la distància acumulada entre els dos segments (D1+D2) i la distància en línia recte entre els punts inicial d'un segment i final de l'altre (D3).



Totes les propietats anteriors menys la variança de la distància són comunes per cadascun dels punts del carrer. Llavors un cop calculades, s'afegeix a cadascun dels nodes Intersecció del carrer. Com que totes les interseccions com a mínim formen part de dos carrers, tenim que cadascuna d'elles té com a mínim dos valors per aquestes propietats. Per això, un cop s'han calculat per a tots els carrers, s'accedeix intersecció a intersecció i és fa la mitjana d'aquestes propietats.
De la propietat variança de la distància no fa falta fer-ne cap mitjana ja que el càlcul es fa intersecció a intersecció i és un sol valor.
Les interseccions també guarden una llista amb el nivell de tots els carrers que les creuen. Aquest nivell bé determinat pel tipus de carrer que el mapa OSM ens indica que és, per més informació sobre els tipus de carrer dels mapes OSM, mirar l'annex1. Per veure la relació entre el nivell i el tipus de carrer mirar l'apartat 4.3.5

3.2 Processament de l'estructura

L'objectiu d'aquest processament és generar un nova xarxa de carrers que sigui semblant a la dels exemples i que a més estigui connectada a aquests.

Per aconseguir aquest objectius s'han seguit els passos següents:

- Seleccionar un conjunt d'interseccions.

- Moure/copiar les interseccions a una area buida.

- Connectar aquests punts considerant els seus atributs calculats anteriorment.

Les interseccions es guarden en un diccionari amb l'identificador com a índex i les seves propietats com les dades associades a aquest índex.
Per facilitat el recorregut per nivells de les interseccions, es crea un altre diccionari amb tots els identificadors de les interseccions classificades per nivell; una intersecció estarà a un o més nivells.

3.2.1 Generació dels carrers

Els carrers es generen unint les interseccions mitjançant un algorisme de camins aleatoris. Un pas en aquest algorisme consisteix en moure's d'una Intersecció d'origen v_i a una altre v_j. Primer s'uneixen les interseccions de major nivell (Carrers més importants) i quan no se'n poden generar més es baixa un nivell.

Aquest algorisme bàsicament el que fa és calcular la probabilitat que té tot el conjunt d'interseccions d'un nivell d'aparellar-se amb la intersecció actual per formar un segment. Hi ha una sèrie de casos en que la probabilitat serà canviada a 0 ja que són combinacions que no es poden permetre, aquest casos són:

- El segment candidat creua amb un segment ja existent.

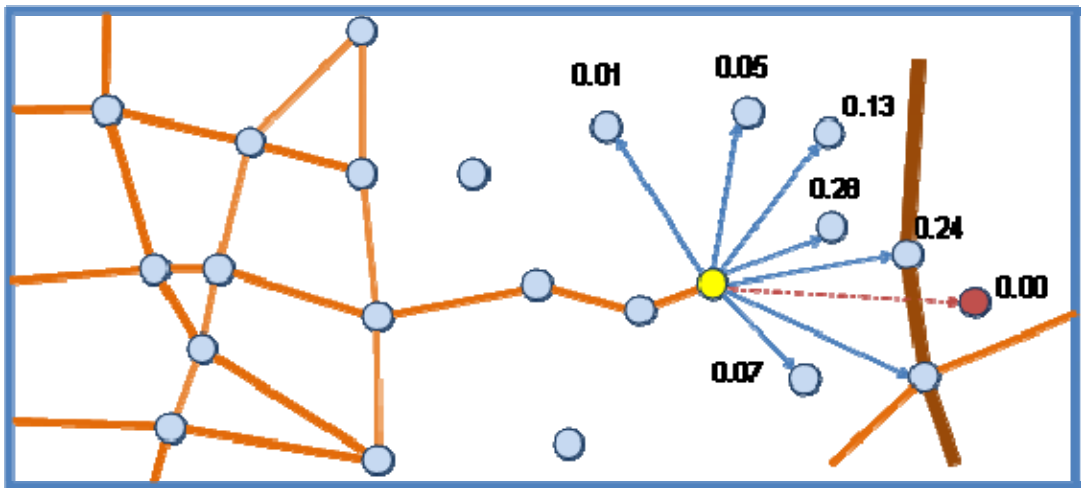


Figura 22: Exemple on es mostra un punt invalid per creauar una aresta ja existent.

- La intersecció no admet més segments (màxim 4)

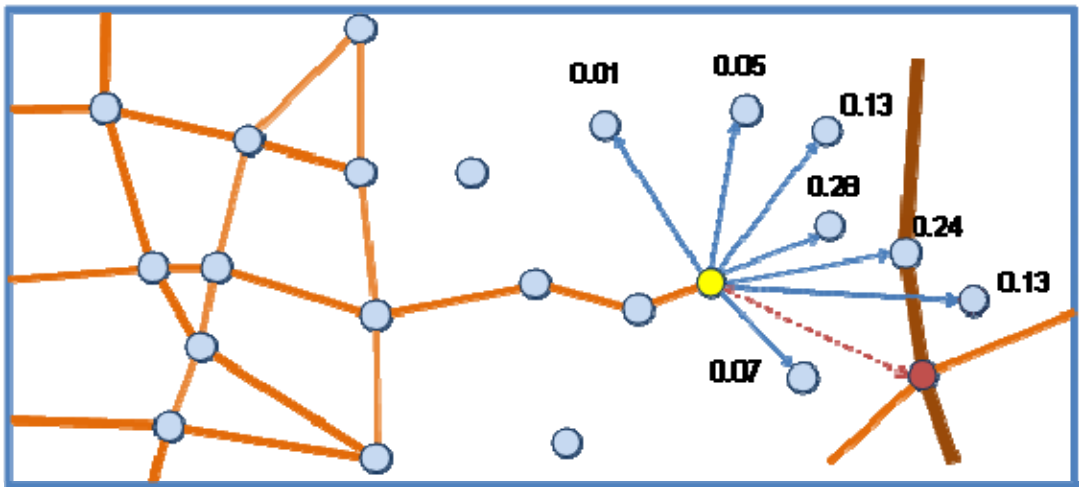


Figura 23: Exemple on es mostra un punt invalid per ja tenir el nombre màxim de segments possibles.

- La transició és excessivament estranya (angle o distància massa gran o massa petita)

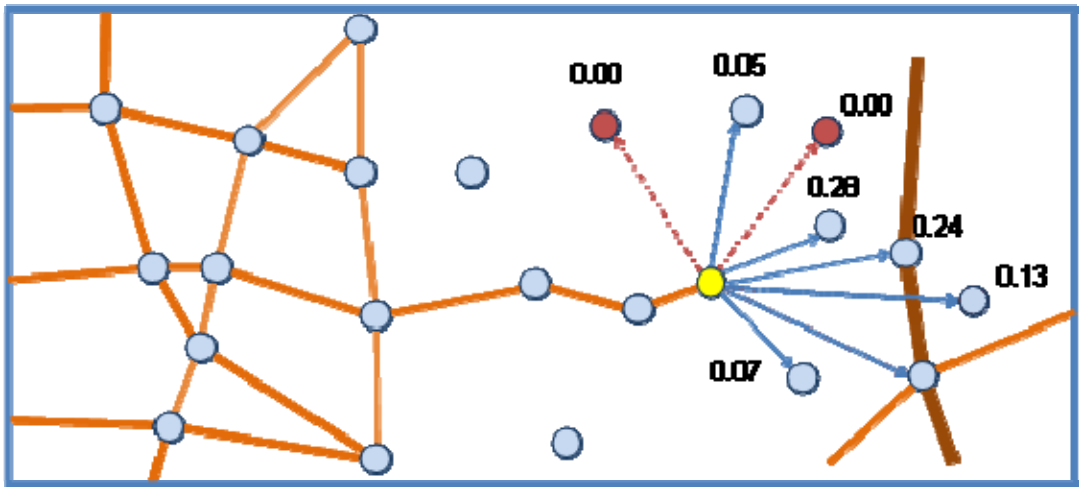


Figura 24: Exemple on es mostren punts amb angles invalids.

- Si un dels punts és la intersecció anterior

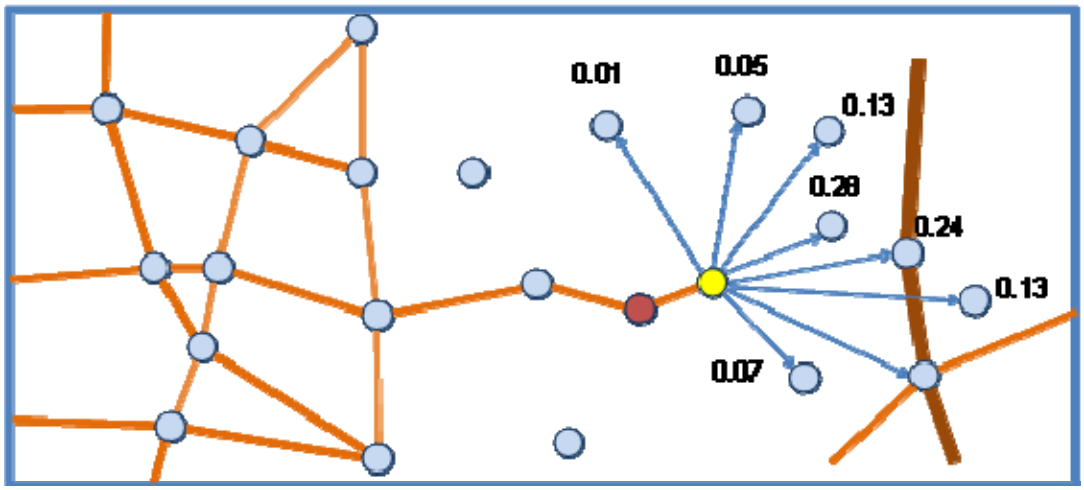


Figura 25: Exemple on es mostra un punt invalid per ser la intersecció anterior.

Per a calcular la probabilitat s'utilitza una funció gaussiana de grau 2 acotada a un valor entre 0 i 1. Quan s'ha finalitzat de calcular totes les probabilitats, es descarten els punts amb probabilitat 0, es tria el punt següent utilitzant una funció cdf. Les dos funcions presentades aquí seran explicades en els subapartats següents.

L'algorisme finalitza quan totes les probabilitats de transició són 0, un cop ha passat això es comença un altre "Camí aleatori" escollint com a inici una Intersecció amb el mateix nivell que l'últim carrer creat que tingui connexions possibles. Si no es troba cap punt que ho compleixi es passa al següent nivell. Un cop s'ha passat per tots els nivell el procés s'acaba.

Per a més informació sobre aquest algorisme accedir a l'apartat 4.2.11, al mètode randomWalk.

3.2.1.1 Funció Gaussiana

La funció gaussiana és una funció definida por l'expressió:

$$f(x) = ae^{-(x-b)^2/2c^2}$$

on a, b i c son constants reals (a > 0).
La gràfica de la funció es simètrica amb forma de campana, coneguda com a campana de Gauss, veure figura 26. El paràmetre a es l'altura de la campana centrada en el punt b, mentres que c en determina l'amplada.

Les funcions gaussianes s'utilitzen sovint en estadística ja que en el cas que a sigui igual a $1/c\sqrt{2\pi}$, representen la funció de densitat d'una variable aleatòria amb distribución normal de mitjana μ=b y variança $\sigma^2=c^2$.

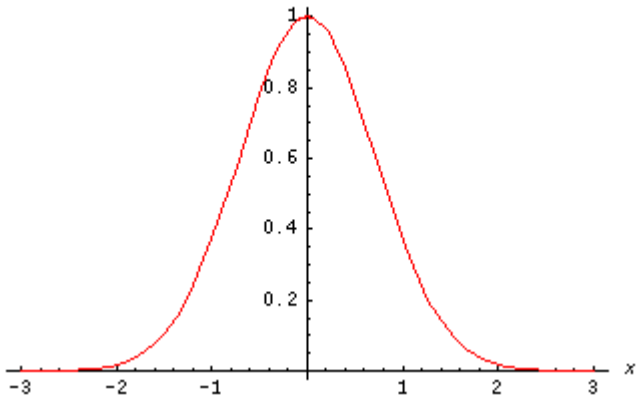


Figura 26: Gràfica d'una funció Gaussiana de grau 1

En aquesta aplicació s'utilitza una gaussiana de grau dos el que significa que la funció en comptes de tenir una variable en té dues. La forma en la que s'ha utilitzat és la següent:

$$f(x,y) = e^{-\left[(x-x_0)^2/(2\sigma_x^2)+(y-x_0)^2/(2\sigma_y^2)\right]}$$

3.2.1.2 Funció de distribució acumulada (CDF)

La funció de distribució F(X), també anomenada funció de distribució acumulada (CDF) o funció de freqüència acumulada, descriu la probabilitat que una variable aleatòria X prengui un valor menor o igual que un nombre Y.

Si X es una variable aleatòria, llavors per a qualsevol nombre real x_0 , existeix una probabilitat $P(X \leq x_0)$ de l'event $X \leq x_0$.

La probabilitat $P(X \leq x_0)$ que depèn de l'elecció de x_0 és la probabilitat acumulada fins a x_0, que es denota coma a $F(x_0) = P(X \leq x_0)$
La taula següent mostra un petit exemple del que estem parlant.

X	f(X)	F(X)
1	0.12	0.12
2	0.344	0.464
3	0.4	0.864
4	0.05	0.869

La gràfica de la funció de distribució acumulada d'una variable discreta es sempre una gràfica escalonada.

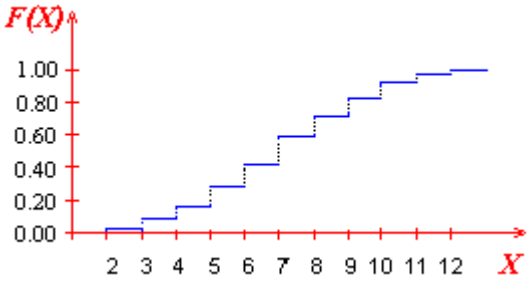


Figura 27 Gràfica de la funció de distribució d'una funció Gaussiana de grau 1

En aquesta aplicació s'utilitza construint la taula igual que en l'exemple on X és l'identificador de cada intersecció i f(X) és el resultat de calcular la gaussiana de grau 2 per aquesta intersecció. Un cop s'ha creat la taula, s'ordena de major a menor f(X), i seguidament és calcula l'acumulació de la probabilitat $F(x_0)$. Una vegada la taula es troba totalment acabada, es genera un nombre aleatori entre 0 i el valor màxim de $F(x_0)$. La intersecció escollida és la primera que té una $F(x_0)$ superior al nombre aleatori generat. A continuació es mostra un petit exemple d'aquest procés.

- Taula d'entrada

X	f(X)
1	0.12
2	0.344
3	0.4
4	0.05

- Taula ordenada

X	f(X)
3	0.4
2	0.344
1	0.12
4	0.05

- Càlcul de la l'acumulació de la probabilitat $F(x_0)$

X	f(X)	F(X)
3	0.4	0.4
2	0.344	0.744
1	0.12	0.864
4	0.05	0.869

- Generació del nombre aleatoi 0.75

- La intersecció final a retorna és la 1 ja que amb **F(X)**=0.864, és la primera amb **F(X)**>0.75

4- Anàlisi

L'objectiu d'aquesta etapa del projecte és investigar el problema a resoldre (que), sense ocupar-se en trobar-ne una solució (com). Durant aquest procés es realitzarà una mena de traducció dels requeriments comentats en el capítol anterior a un llenguatge més formal. En definitiva el propòsit de l'anàlisi orientat a objectes consisteix en definir l'anomena't *model del domini*, és a dir, totes les classes que seran necessàries per resoldre el problema, així com els atributs i operacions associades a cadascuna de les classes.

4.1 Requeriments del Sistema

Els requeriments del sistema recullen els principals objectius de l'aplicació juntament amb les accions que haurien de poder dur a terme cadascun dels usuaris.

Els requeriments del sistema ens permeten identificar els elements que envoltaran el sistema:

- Les persones o elements que interactuaran amb el sistema (Actors).
- Les funcionalitats que ha d'oferir el sistema.
- El hardware o plataforma sobre la que funcionarà el sistema.
- El software que s'utilitzarà a l'hora d'implementar el sistema.
- Existeixen altres elements com ara pressupostos, Gestors de bases de dades, etc. Que en aquest cas no afecten al sistema que s'està implementant.

Hi ha dos tipus de requeriments:

- **Funcionals**: Els requeriments funcionals són aquells que denoten una funcionalitat o servei que tindrà l'aplicació.
- **No Funcionals**: Els requeriments no funcionals són les restriccions que ens venen imposats des del client o de la pròpia naturalesa del problema.

4.1.1 Requeriments funcionals

A continuació es descriuran i classificaran els principals requeriments funcionals que compleix aquesta aplicació:

- Carregar mapes OSM

  Una de les principals funcionalitats que ha de complir l'aplicació és poder carregar un mapa (exemple) en format OSM.

- Generar Xarxa Carrers

  Es tracta de la funció bàsica de l'aplicació, en la que s'agafa l'exemple carregat anteriorment, se'n selecciona un troç o tot ell i seguint uns paràmetres modificables per l'usuari és genera una xarxa de carrers, un cop generada l'aplicació dona l'opció de visualitzar-lo, guardar-lo o generar una ciutat (Vista 3D)

- Seleccionar Area

  L'usuari pot seleccionar el troç de ciutat que vol fer servir per generar la nova xarxa de carrers

- Escollir Paràmetres

  L'usuari pot escollir una sèrie de paràmetres a l'hora de generar la nova xarxa de carrers, com ara l'angle mínim i màxim entre dos segments del mateix carrer, la distància mínima entre dos carrers diferents,...

- Guardar Mapa

  L'aplicació permet guardar el mapa de carrers generat en format OSM.

- Generar ciutat

  L'aplicació genera una ciutat en 3D agafant com a base la xarxa de carrers generada

- Visualitzar

L'usuari pot visualitzar tan el mapa original com la xarxa de carrers en 2D

4.1.1.1 Diagrames de casos d'ús

Els casos d'ús ofereixen una descripció funcional d'un sistema i dels seus principals processos, i delimita el problema a tractar. També ofereixen una descripció gràfica de qui utilitzarà el sistema.

Un cas d'ús és un patró del comportament del sistema, que es representa com una seqüència de transaccions relacionades que són executades per un actor i el sistema en un diàleg.

- Diagrama de cas d'ús general

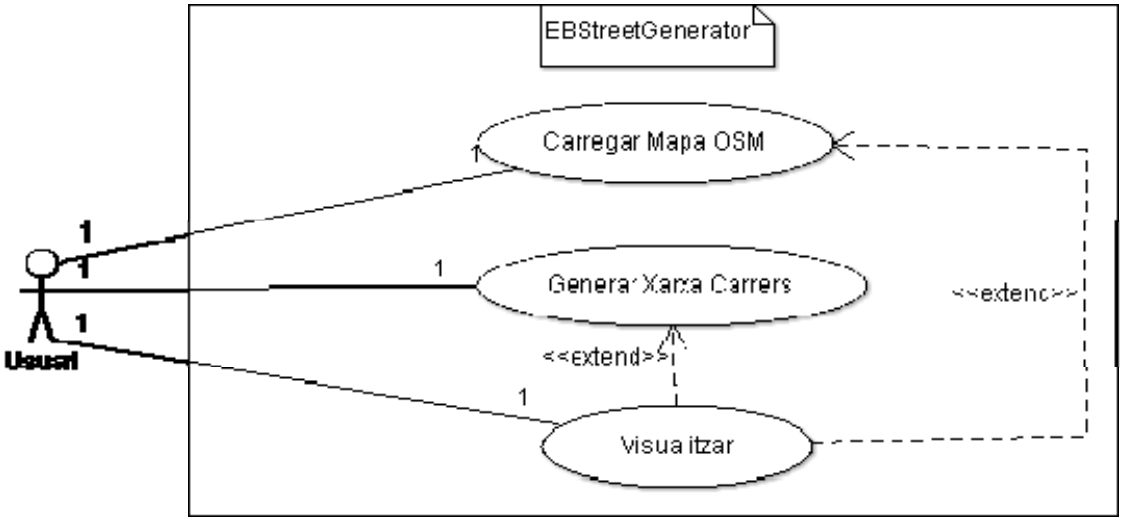


Figura 28: Diagrama de cas d'ús general

- Diagrama de cas d'ús Carregar mapa OSM

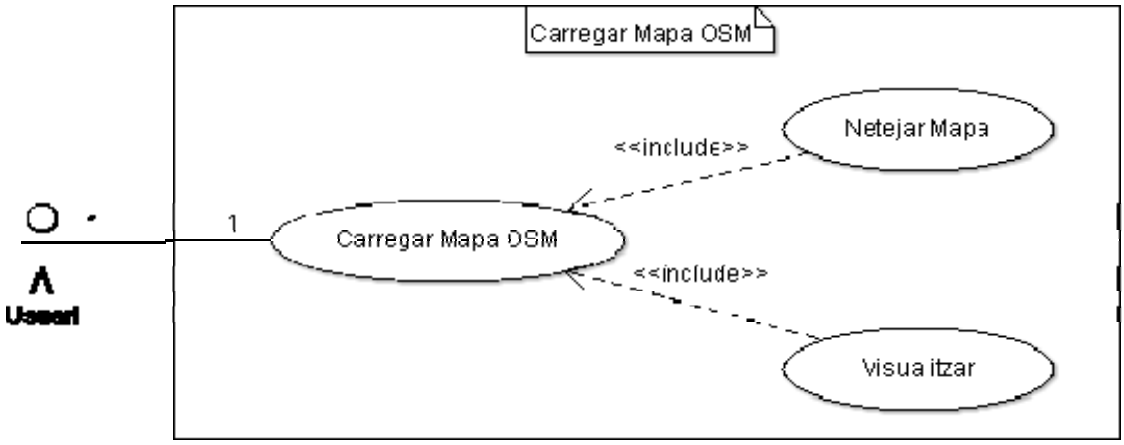


Figura 29: Diagrama de cas d'ús Carregar mapa OSM

- Diagrama de cas d'ús Generar Xarxa Carrers

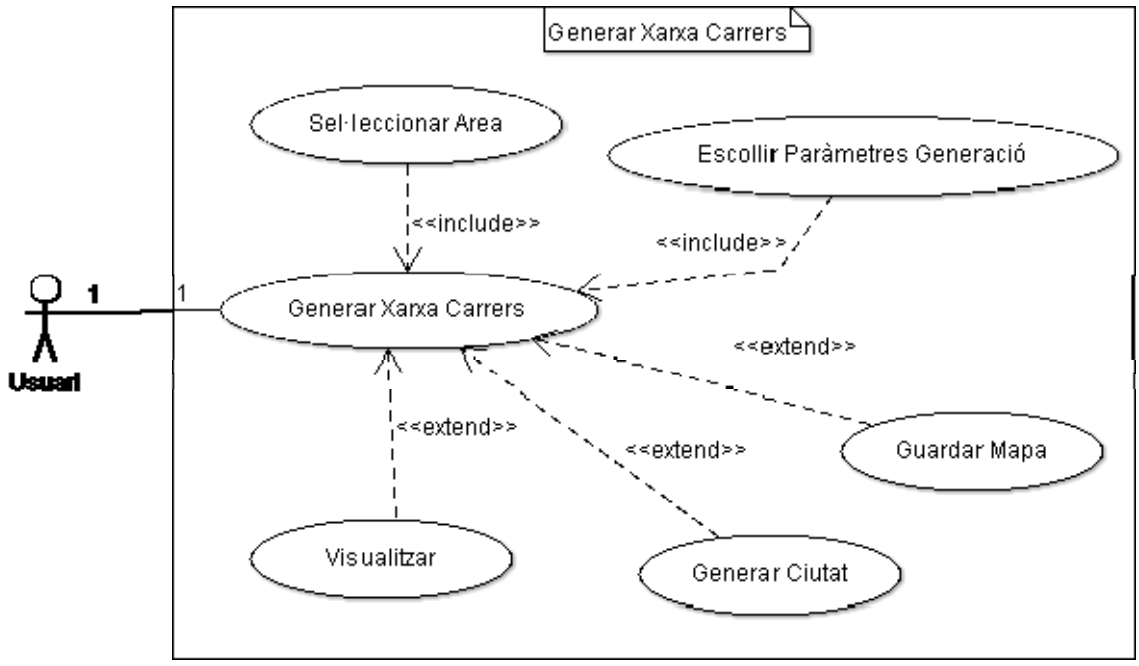


Figura 30: Diagrama de cas d'ús Generar Xarxa de Carrers

- Diagrama de cas d'ús Escollir Paràmetres

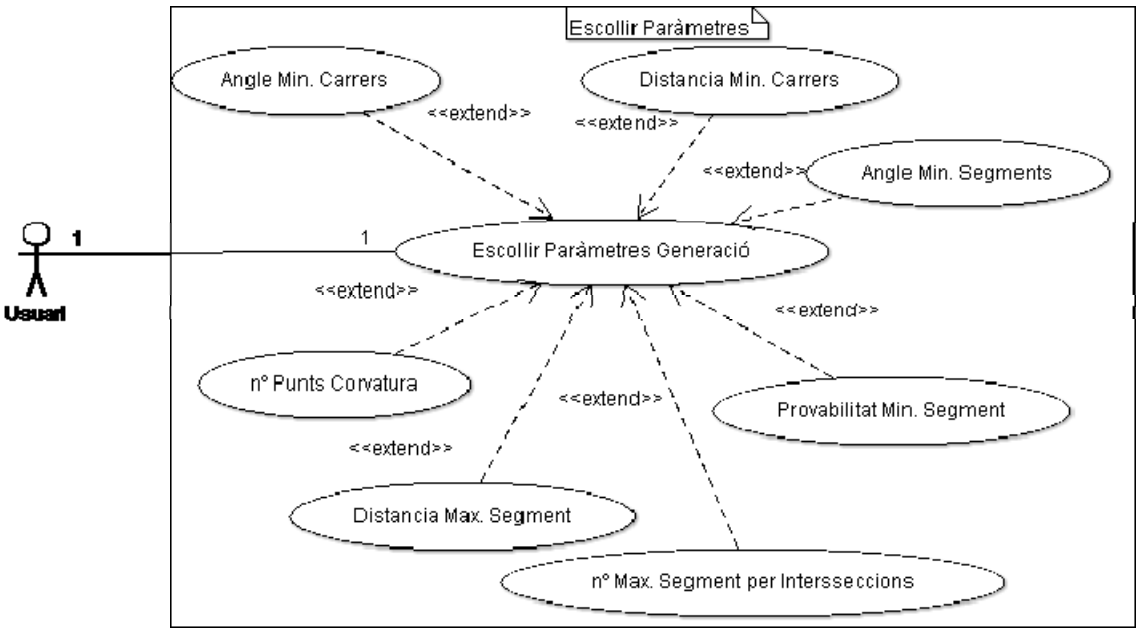


Figura 31: Diagrama de cas d'ús Escollir Paràmetres

- Diagrama de cas d'ús Visualitzar

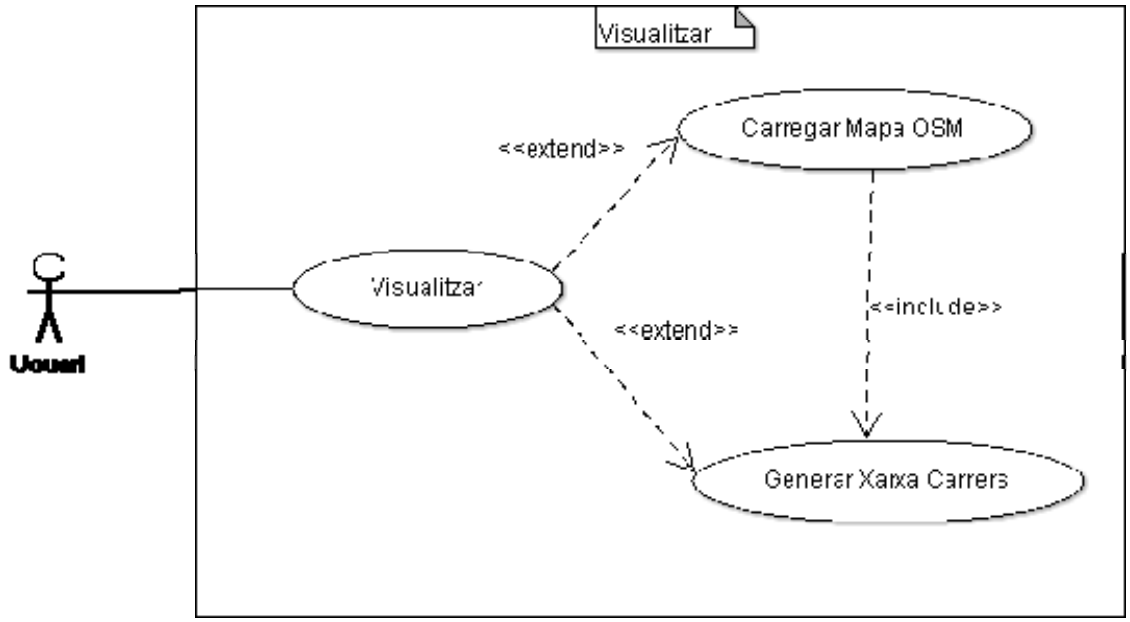


Figura 32: Diagrama de cas d'ús Visualitzar

4.1.1.2 Fitxes dels casos d'ús

- Fitxa cas d'ús Carregar mapa OSM

FITXA	Carregar mapa OSM
Funcionalitat	Es carrega el Mapa OSM sobre el que es treballarà
Actor(s)	Usuari.
Pre-condició	cap
Flux principal	1- L'usuari entra el path del mapa OSM que s'utilitzarà com a Exemple. 2- El sistema elimina les dades no necessàries del mapa 3- Es visualitza el mapa
Subfluxos (extensions)	Cap
Flux alternatiu	Cap
Post-condició	Cap

- Fitxa cas d'ús Generar Xarxa Carrers

FITXA	Generar Xarxa Carrers
Funcionalitat	Genera una nova xarxa de carrers a partir d'un tall de l'exemple carregat
Actor(s)	Usuari.
Pre-condició	Tenir carregat un Mapa OSM
Flux principal	1- Seleccionar l'àrea a partir de la qual es crearà la nova xarxa. 2- Es Seleccionen els Paràmetres de la generació.
Subfluxos (extensions)	o Es dibuixa la xarxa de carrers en 2D o Es guarda la xarxa de carrers generada com un mapa OSM o Es genera i mostra una ciutat en 3D a partir de la xarxa de carrers.
Flux alternatiu	Cap
Post-condició	Cap

- Fitxa cas d'ús Escollir Paràmetres

FITXA	Escollir Paràmetres
Funcionalitat	Es modifiquen els paràmetres que s'utilitzaran per a la generació de la xarxa de carrers
Actor(s)	Usuari.
Pre-condició	Cap

Flux principal	1- L'usuari pot establir: • L'angle mínim entres carrers • La distància mínima entre carrers • L'angle mínim entre segments • El nombre màxim de segments que poden passar per cada intersecció • La probabilitat mínima del segment generat per ser vàlid • La longitud màxima dels segments i el nº de punts de curvatura.
Subfluxos (extensions)	Cap
Flux alternatiu	Cap
Post-condició	Cap

- Fitxa cas d'ús Visualitzar

FITXA	Escollir Paràmetres
Funcionalitat	Es visualitza el mapa carregat o la xarxa de carrers generada segons el moment.
Actor(s)	Usuari.
Pre-condició	Tenir carregat un Mapa OSM
Flux principal	1- Es carrega un mapa OSM i seguidament aquest es mostra.
Subfluxos (extensions)	cap
Flux alternatiu	1- Es genera una xarxa de carrers i seguidament el sistema esborra el mapa OSM que s'està mostrant i mostra la xarxa de carrers.
Post-condició	cap

4.1.2 Requeriments no funcionals

Aquest projecte ha estat plantejat de tal manera que tingui el menor nombre de requeriments no funcionals possibles. El fet de ser una aplicació monousuari comporta que no hi hagi necessari utilitzar cap tipus de sistema de seguretat, i les eines que utilitza s'ha intentat que fossin el menys restrictives possibles.

Els requeriments d'aquest projecte són els mateixos que els necessaris perquè funcioni el Houdini 10 (amb les versions 9 també funciona però d'una manera poc estable):

- Sistema Operatiu
  - Windows: Windows XP i Vista (32 i64 bit)
  - Mac OS X: OS X v10.5 Leopard en un MAC basat en processadors Intel
  - Linux:
    - Ubuntu 7.10
    - Debian 3.1 and 4.0 (32 and 64-bit)
    - RHEL 4
    - Fedora Core 6

  Es necessiten drets d'administració per a poder instal·lar el Houdini
  Els escriptoris amb el GL activat encara no estan suportats

- Memòria
  - Es necessiten un mínim de 2GB
  - Es recomanen de 2 a 4 GB

- Processador
  - Processador AMD o Intel de 32 o 64 bits que tinguin instruccions SSE o superior

  El Houdini suporta les instruccions MMX i les (SSE2) si estan disponibles
  El Houdini suporta multiprocessament per composar i renderitzar

- Espai al Disc
  - Es necessita un mínim de 500 MB lliures.

- Dispositius d'entrada
  - Es requereix ratolí de 3 botons

- Targeta gràfica
  - Targeta gràfica NVidia o ATI que suporti OpenGL 1.2 o superior.
  - Versió Drives de la targeta gràfica: NVidia: 169 o superior, ATI: 8.44 o superior.
  - Resolució mínima: 1024 x 768, es recomana 1920 x 1200 (Panoràmica).
  - Les targetes integrades Intel no estan suportades.

4.2 Diagrames d'activitat

Els diagrames d'activitat presenten una seqüència d'"activitats" i com el camí d'execució va d'una a l'altra, s'organitzen respecte a les accions i principalment estan destinats a representar el comportament intern d'un mètode (la realització d'una operació), d'un cas d'us o d'un procés de negoci (Workflow)

Quan una activitat acaba es desencadena el pas a la següent activitat. Les activitats no tenen transicions internes ni transicions desencadenades per events.

Els diagrames d'activitat també poden tenir:

- Punts de decisió: condicions que canvien el flux de les accions
- Camins d'execució paral·lels
- Senyals, amb punts d'enviament i recepció explícits
- Objectes afectats per les accions

4.2.1 Diagrama d'activitat Carregar Mapa OSM

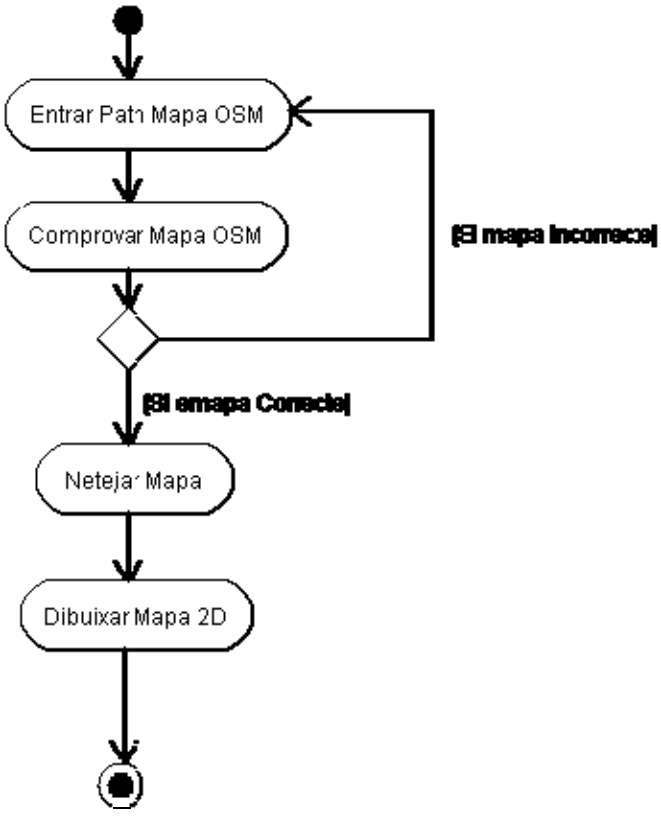


Figura 33: Diagrama d'activitat Carregar Mapa OSM

En aquest diagrama es mostren els passos que és segueixen per carregar mapes OSM. Aquest passos consisteixen en que l'usuari entri el path del mapa OSM. Un cop entrat, el sistema comprova que el fitxer existeix i que no tingui errors, si no passa aquest procés es demanarà a l'usuari que entri el path d'un altre mapa. Si el sistema no troba cap error al mapa, s'eliminen tots els elements que tingui excepte els nodes i els carrers i seguidament és dibuixen els carrers al Houdini.

4.2.2 Diagrama d'activitat seleccionar àrea

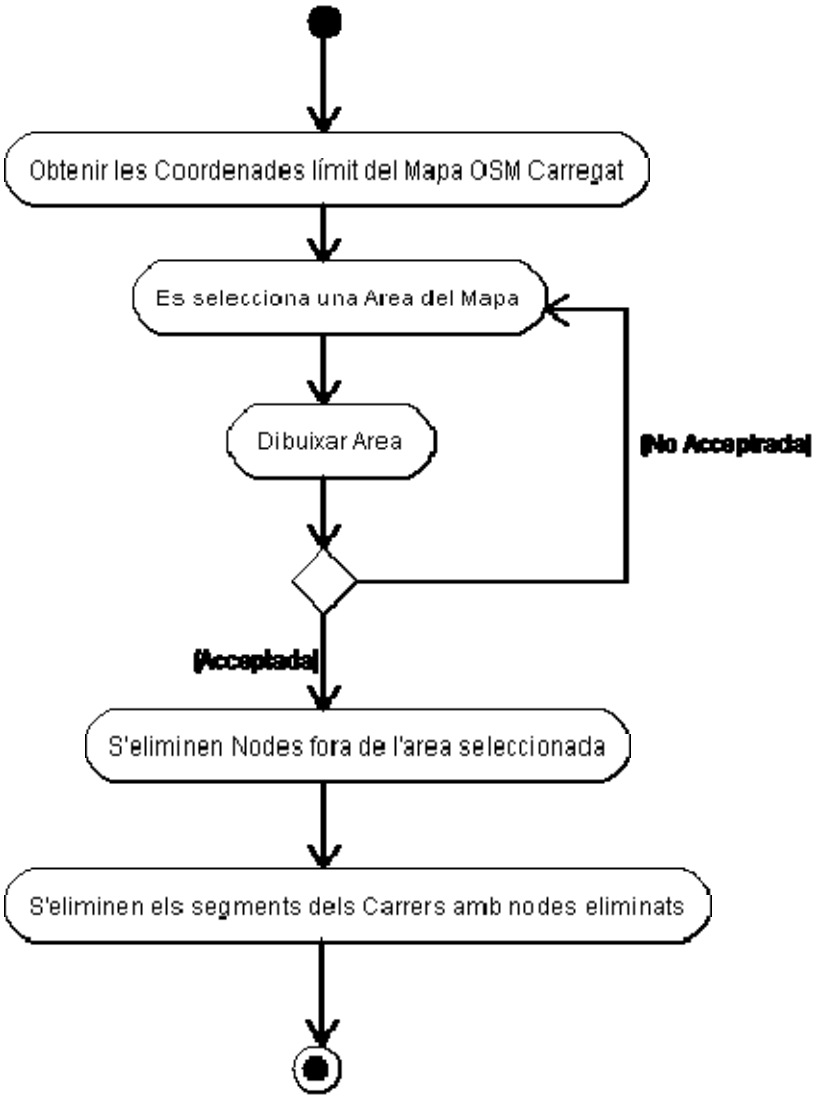


Figura 34: Diagrama d'activitat seleccionar àrea

En aquest diagrama es mostra el procediment que es segueix per seleccionar un tall o tot el mapa que acabem de carregar. Aquest procediment consisteix en obtenir la latitud i longitud mínima i màxima del mapa per tal de poder-les presentar a l'usuari com a límit perquè no escolli coordenades externes al mapa. Un cop fet això l'usuari ha d'escollir una àrea del mapa (pot ser tot ell), quan ja l'ha escollit es dibuixen els carrers que conté aquesta area al Houdini. Si a l'usuari li agrada l'àrea escollida, li comunica al programa que l'accepta i aquest elimina els nodes que és troben fora d'aquesta area i els segments dels carrers que tenien algun d'aquests nodes. Si no pot tornar a repetir el procés fins que en trobi una el que satisfaci

4.2.3 Diagrama d'activitat visualitzar mapa

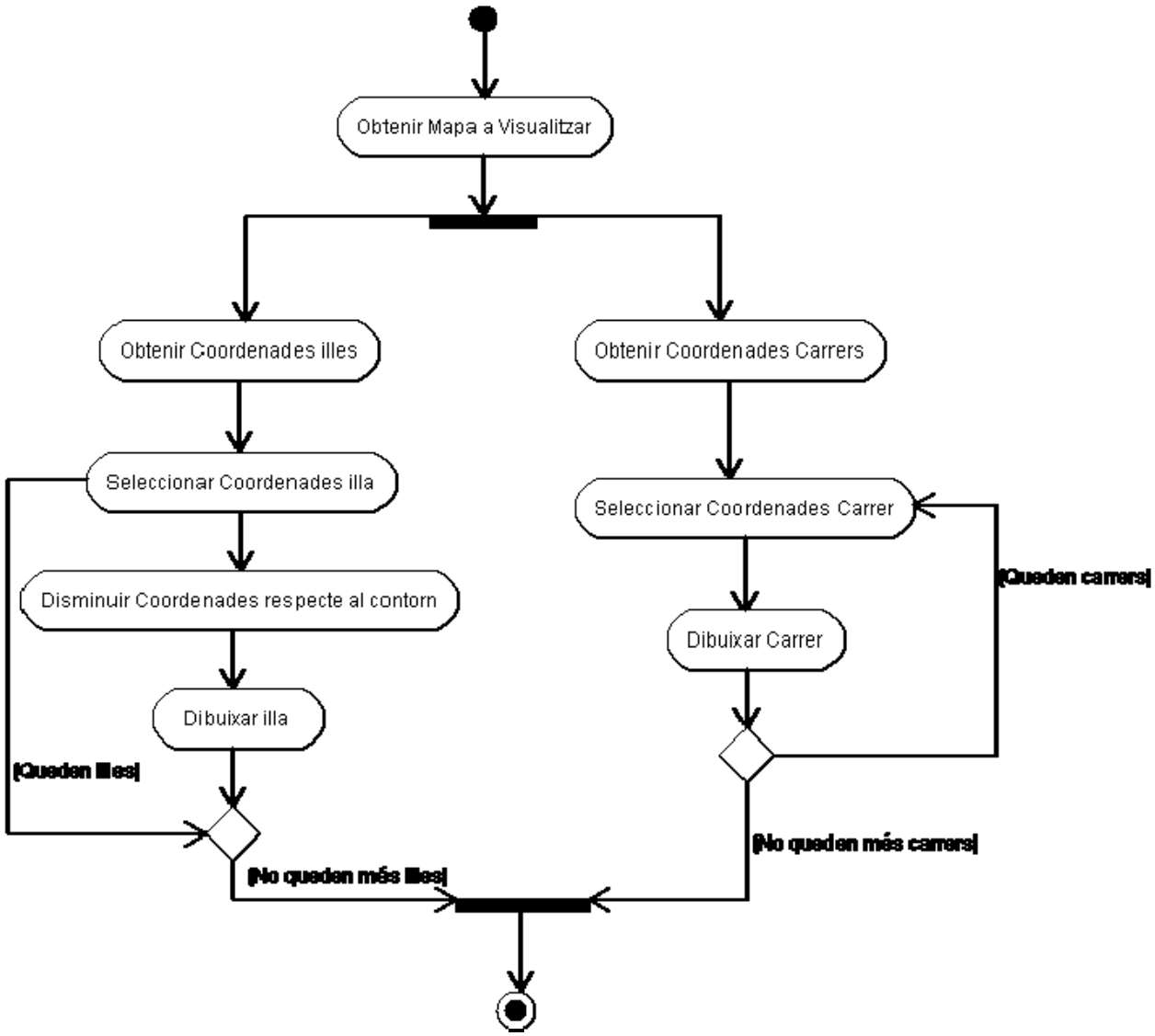


Figura 35: Diagrama d'activitat visualitzar mapa

En aquest diagrama es mostra les dues opcions que es donen al dibuixar. El primer que es fa és aconseguir el mapa que es desitja dibuixar. Després l'usuari escull si vol que és dibuixin els carrers o les illes. Si escull les illes el següent és aconseguir per a cadascuna de illes les coordenades de tots els punts que formen cadascuna d'elles. Un cop es tenen aquestes coordenades es procedeix a seleccionar les coordenades de cada illa i es disminueixen un mica per tal de donar una certa amplada als carrers que les delimiten. Un cop s'han disminuït ja es poden dibuixar.

En el cas que l'usuari hagués escollit dibuixar els carrers el procediment començaria aconseguint les coordenades de tots els punts que formen cadascun dels carrers. Seguidament s'escollirien les coordenades que formen cada carrer i es dibuixarien.

4.2.4 Diagrama d'activitat generar xarxa de carrers

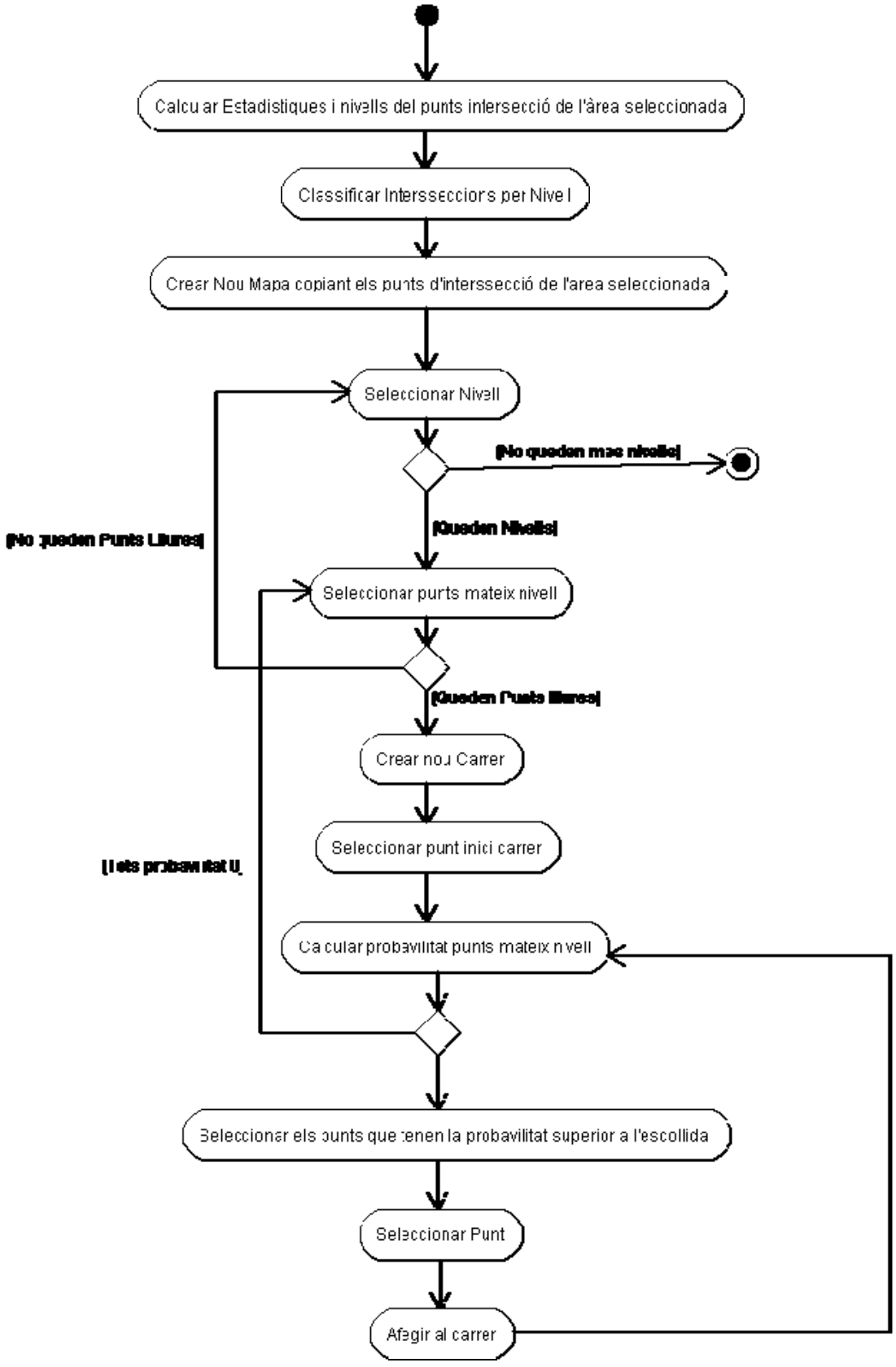


Figura 36: Diagrama d'activitat generar xarxa de carrers

En aquest diagrama es mostra el procediment que es segueix per poder generar una xarxa de carrers. Es parteix del supòsit que es té un mapa OSM carregat i un àrea seleccionada. Per començar el programa ha de calcular totes les estadístiques dels carrers i guardar-les a cadascuna de les interseccions que els formen. Un cop s'ha finalitzat aquest procés es procedeix a classificar les interseccions segons els seus nivells i es copien a un nou mapa. Seguidament es procedeix a la creació dels nous carrers. Per fer-ho és seleccionen tots els punts del nivell més alt. D'entre aquests se'n selecciona un que representa el punt inicial del carrer. Seguidament es calcula la probabilitat que tenen totes les interseccions del mateix nivell de formar un segment amb el punt inicial. Un cop les hem calculat es desprecien les que siguin menors a un cert nombre (escollit per l'usuari anteriorment) i amb les que queden se n'escull una, s'afegeix al carrer i es repeteix el mateix procediment que amb el cas del punt inicial fins que el sistema es troba que totes les probabilitats per a la intersecció actual són 0. Quan passa això es dona per finalitzat el carrer i s'inicia la creació d'un altre carrer del mateix nivell que l'anterior. Quan no es poden crear més carrers del mateix nivell es passa al següent nivell fins que ja no en queden més. Quan passa això la creació de la xarxa de carrers ha finalitzat.

4.3 Diagrama de classes

Un diagrama de classes mostra l'existència de les classes i les seves relacions dins de la vista lògica d'un sistema.

Elements que formen els diagrames de classe:

- Classes amb la seva estructura i comportament
- Relacions:
  - Associació
    Una relació associació és una connexió bidireccional entre classes
  - Agregació

    Una agregació és una relació més forta on s'estableix un lligam entre una entitat i les seves "parts"
  - Dependència

    Una relació de dependència és un tipus de relació més feble que mostra una relació entre un client i un proveïdor en la que el client no té coneixement semàntic del proveïdor
  - Herència

    L'herència és una relació entre una superclasse i les seves sub-classes. Hi ha dues formes de trobar herència:
    - Generalització
    - Especialització
- Indicadors de multiplicitat i navegació
- Nom de "rols"

Una classe és una col·lecció d'objectes amb una estructura, comportament, relacions i semàntica comú. Les classes es codifiquen seguint:

- Una classe es representa com un rectangle amb tres compartiments.
- Les classes s'ha d'etiquetar utilitzant paraules del domini concret
- S'han de crear estàndards de noms (ex. Totes les classes seran noms singulars que comencen amb majúscula)

4.3.1 Diagrama de classes general

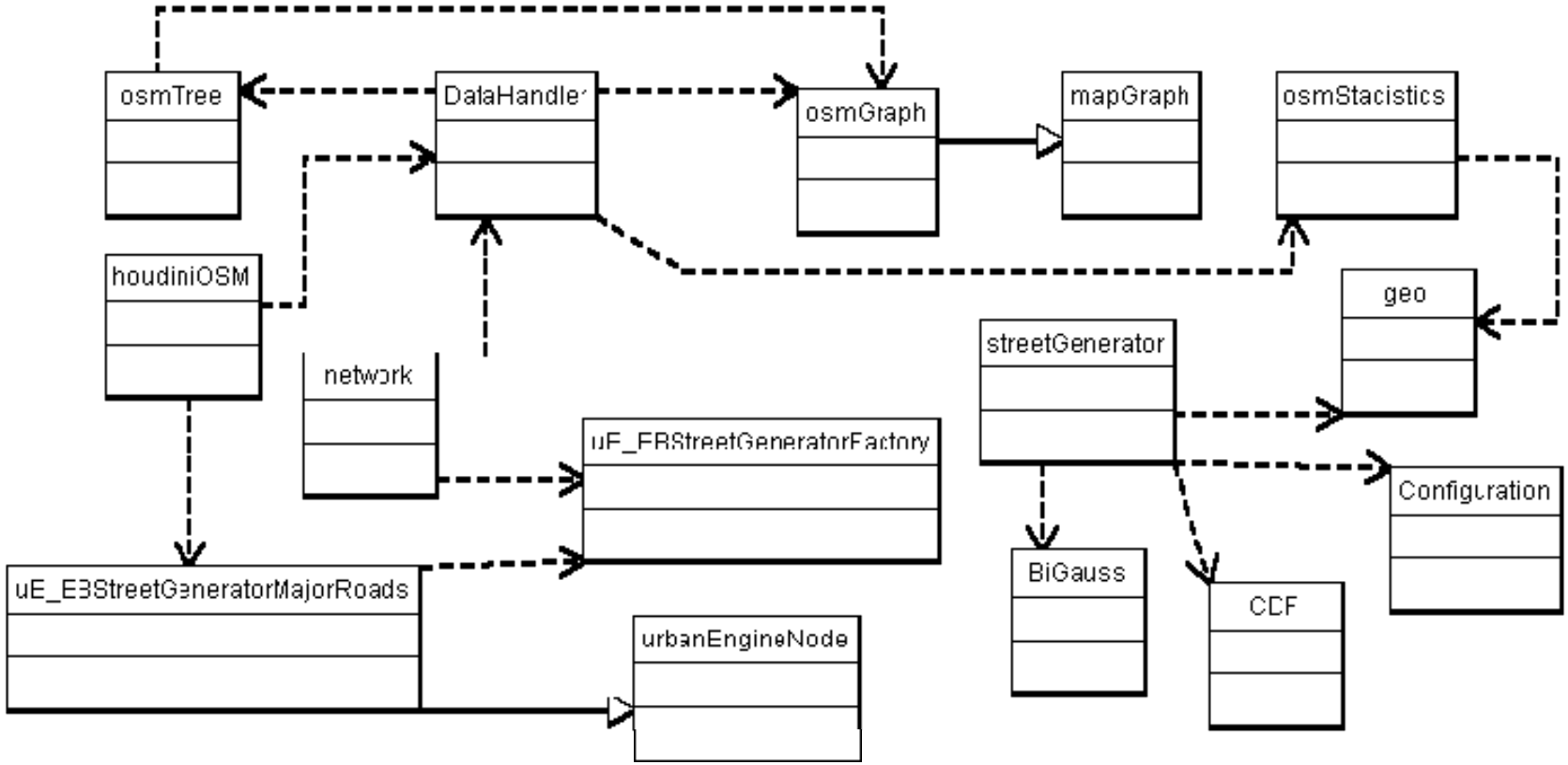


Figura 37: Diagrama de classes general

Aquest diagrama representa la distribució de les classes d'aquesta aplicació així com les seves interrelacions. Tot aquest conjunt permet descriure totes les operacions definides en la fase de requeriments i representades esquemàticament en forma de paquets en l'apartat anterior.

Com es pot apreciar en el diagrama, les interrelacions entre classes d'aquesta aplicació són del tipus més feble que hi ha, això és així ja que a l'hora de dissenya-la es va creure que d'aquesta manera seria més fàcil reutilitzar el codi en un futur.

4.3.2 Classe osmTree

Aquesta classe converteix els fitxers OSM en arbres miniDOM i viceversa; també ofereix una sèrie de mètodes per poder treballar sobre l'arbre miniDOM.
Cada objecte d'aquesta classe guarda el seu arbre miniDOM a l'atribut root.

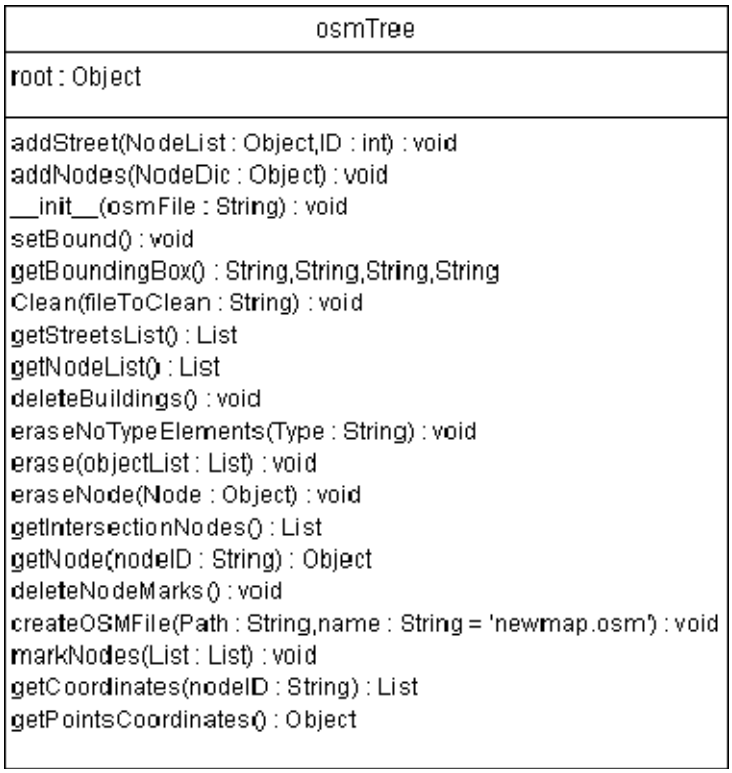


Figura 38: Classe osmTree

- __init__(osmFile)

  Aquest és el mètode constructor. A partir d'una ruta a un mapa OSM (*osmfile*), genera un objecte d'aquesta classe i assigna a root l'arbre miniDOM resultant de parsejar el mapa OSM. Si no se li passa cap ruta, genera un arbre OSM que només conté la capçalera dels mapes OSM (per més informació mirar l'apartat 2.3.3.1).

- addStreet(NodeList,ID)

  Aquest mètode crea un element de tipus way (més informació a l'annex1) amb la id (*ID*) especificada i tants fills del tipus nd com nodes hi ha a la llista *NodeList*, assignant a cada node una id diferent de les que es troben dins de *NodeList*.

L'element creat s'afegeix a l'arbre OSM a l'apartat de carrers.
Per a més informació de que representen aquests tipus de node, mirar l'apartat 2.3.3.3.

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1  funció addStreet(NodeList,ID):
 2      osm:= Primer fill de root
 3      way:= Crear nou element del tipus way
 4      osm.AfegirFill(way)
 5      way.setAttribute("id",ID)
 6      way.setAttribute("visible","true")
 7      per cada node Node de NodeList fer:
 8          node:= Crear nou element del tipus nd
 9          osm.AfegirFill(node)
10          node.setAttribute("ref",Node)
11          way.AfegirFill(node)
12      node2:= Crear nou element del tipus tag
13      osm.AfegirFill(node2)
14      node2.setAttribute("k","name")
15      node2.setAttribute("v",("Street"+str(ID)))
16      way.AfegirFill(node2)
17      node3:= Crear nou element del tipus tag
18      osm.AfegirFill(node3)
19      node3.setAttribute("k","highway")
20      node3.setAttribute("v","residential")
21      way.AfegirFill(node2)
```

- addNodes(NodeDic)

  Aquest mètode rep una diccionari en format {Id1:(Lon1,Lat1), Id2:(Lon2,Lat2),..} (*NodeDic*) i crea un node del tipus node amb id, latitud i longitud iguals a les donades per cadascun dels parells id/valor del diccionari.
  Un cop creats els afegeix a l'arbre miniDOM a l'apartat dels nodes.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1  funció addNodes(NodeDic):
 2      osm:= Primer fill de root
 3      NodeList=llista ids de NodeDic
 4      per cada node element de NodeList fer:
 5          Node=Crear nou element del tipus node
 6          osm.AfegirFill(Node)
 7          Node.setAttribute("id",element)
 8          Lat=valor latitud guardat a NodeDic per a element
 9          Lon=valor longitud guardat a NodeDic per a element
10          Node.setAttribute("lat",Lat)
11          Node.setAttribute("lon",Lon)
```

- setBounds()

  Aquest mètode crea un element de tipus bounds (més informació a l'apartat 2.3.3.1) amb els atributs minlat, minlon, maxlat i maxlon que especifiquen les coordenades mínimes i màximes d'un mapa OSM.
  Un cop creat s'afegeix a l'arbre a la posició que li correspon.

- getBoundingBox ()

  Aquest mètode busca entre els nodes de l'arbre miniDOM els que tenen les coordenades mínimes i màximes i les retorna en l'ordre següent: minLat, minLon, maxLat, maxLon.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1  funció getBoundingBox():
 2       Nodes:=getNodeList()
 3       minLat:=Latitud del primer node de Nodes
 4       minLon:=Longitud del primer node de Nodes
 5       maxLat:=Latitud del primer node de Nodes
 6       maxLon:=Longitud del primer node de Nodes
 7       per cada node node de Nodes fer:
 8           lon=node.getAttribute('lon')
 9           lat=node.getAttribute('lat')
10           si lon major que maxLon llavors:
11               maxLon:=lon
12           altresi lon més petit que minLon llavors:
13               minLon:=lon
14           si lat major que maxLat llavors:
15               maxLat:=lat
16           altresi lat més petit que minLat llavors:
17               minLat:=lat
18       retorna minLat, minLon, maxLat i maxLon
```

- clean(fileToClean)

  Elimina del fitxer que se li dona (*fileToClean*) els salts de línia, els tabuladors i, si hi ha, quatre espais seguits (representació de les tabulacions en alguns editors), i el retorna.

- getStreetsList()

  Retorna un subarbre amb tots els elements de tipus way (per a més informació mirar l'annex1) .

- getNodeList()

  Retorna un subarbre amb tots els elements de tipus node (per a més informació mirar l'apartat 2.3.3.2)

- deleteBuildings()

  Elimina de l'arbre miniDOM tots els elements de tipus tag que tinguin k=building (més informació a l'annex1)

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció deleteBuildings():
 2       ways:=getStreetsList()
 3       per cada carrer way de ways fer:
 4           tags=Tots els elements del tipus tag que tingui way
 5           per cada tag tag de tags fer:
 6               si tag és de tipus building llavors:
 7                   eraseNode(way)
 8                   sortir
```

- eraseNoTypeElements(Type)

  Elimina de l'arbre miniDOM, en l'apartat de carrers, tots els elements de tipus tag que no tinguin k=*Type*, en aquesta aplicació s'utilitza per eliminar totes les vies que no siguin highway (més informació a l'annex1).

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció eraseNoTypeElements(Type):
 2       ways:=getStreetsList()
 3       isHighWay:= Fals
 4       per cada carrer way de ways fer:
 5          tags:=Tots els elements del tipus tag que tingui way
 6          per cada tag tag de tags fer:
 7               si tag és de tipus Type llavors:
 8                   isHighWay:=Cert
 9          si isHighWay llavors:
10               isHighWay:= Fals
11          altrament:
12               eraseNode(way)
```

- erase(objectList)

  Elimina de l'arbre tots els element de dins d'*objectList* .

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció erase(objectList):
 2      father=Nul
 3      per cada node element de objectList fer:
 4          father=element.ObtenirPare()
 5          father.eliminarFill(element)
```

- eraseNode(Node)

  Elimina el node *Node*

- getIntersectionNodes()

  Retorna una llista amb l'identificador de tots els nodes intersecció (node que apareix al llistat de nodes de més d'un carrer).

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció getIntersectionNodes():
 2      streets=getStreetsList()
 3      nodeList:=intersectionList:= Llista Buida
 4      per cada node nodeElement de streets fer:
 5          nodes:=Tots els elements del tipus nd de nodeElement
 6          per cada node node de nodes fer:
 7              ref:= node.getAttribute('ref')
 8              nodeList.Afegir(ref)
 9      per cada node element de nodeList fer:
10          n:=calcular el nº d'aparicions de element a nodeList
11          usat:= es troba dins de intersectionList
12          si n es major a 1 i no usat llavors:
13              intersectionList.Afegir(element)
14      retorna intersectionList
```

- getNode(nodeID)

  Retorna el node amb l'identificador igual a *nodeID*

- deleteNodeMarks()

  Elimina totes les marques que poden tenir els nodes, com per exemple senyals de stop, semàfors,...
  En aquesta aplicació aquest mètode juntament amb el eraseNoTypeElements s'utilitzen per deixar l'arbre només amb els nodes i amb els carrers.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció deleteNodeMarks():
2       NodeL:=getNodeList()
3       Childs:=Llista Buida
4       per cada node element de NodeL fer:
5           childs:= Obtenir els fills de element
6           per cada fill Child de childs fer:
7              element. eliminarFill(child)
```

- CreateOSMFile(Path,name='newmap.osm')

  Genera un mapa OSM a partir de l'arbre miniDOM, el guarda a *Path* amb el nom *name*

- markNodes(List)

  Marca el Nodes que hi ha dins de *List* com si hi hagués un hospital (és la marca que es veu millor).
  Aquest mètode no s'utilitza a l'aplicació final però ha estat molt útil a l'hora de resoldre una multitud d'errors, ja que permet ressaltar la combinació de nodes que és vulgui.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció markNodes(List):
2      per cada node element de List fer:
3          Node=getNode(element)
4          mark= Crear nou element del tipus tag
5          mark.setAttribute('k','amenity')
6          mark.setAttribute('v','hospital')
7          Node.AfegirFill(mark)
```

- getCoordinates(nodeID)

  Retorna les coordenades del node amb l'identificador igual a *nodeID* en format (longitud, latitud).

- getPointsCoordinates()

  Retorna un diccionari amb l'identificador de cada node com a índex i les seves coordenades en format (longitud, latitud) com a dades.

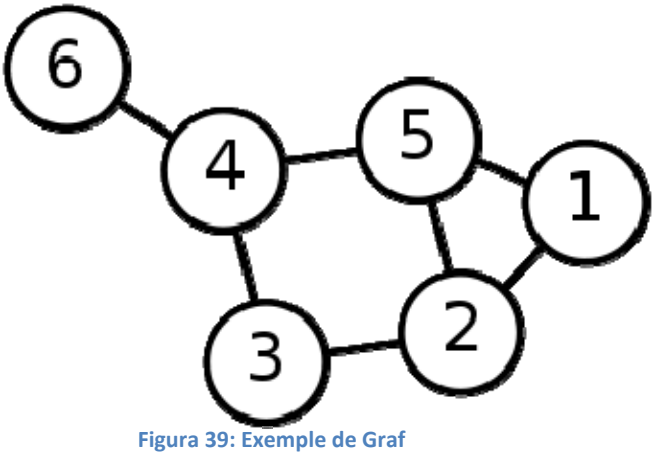
  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció getPointsCoordinates(self):
2      nodes:=getNodeList()
3      coordinates:=Diccionari buit
4      per cada node node de nodes fer:
5          ID:=node.getAttribute('id')
6          Lon:= node.getAttribute('lon')
7          Lat:= node.getAttribute('lat')
8          coordinates[ID]=Tupla(Lon,Lat)
9      retorna coordinates
```

4.3.3 Classe mapGraph

Aquesta classe representa un graf genèric que s'utilitza com a base per definir la classe osmGraph (mirar següent apartat). En aquesta aplicació s'utilitza com si fos una classe abstracta tot i que en altres aplicacions es podria utilitzar directament ja que té tots els mètodes implementats. Aquesta classe s'ha creat per poder definir una estructura de dades que ajudi a simplificar moltes de les funcions que s'utilitzen, ja que implementar-les s'obre la classe osmTree resultaria molt més complicat i costós.

Cada objecte d'aquesta classe guarda un diccionari anomenat Graph que emmagatzema per cadascuna de les ids de les interseccions una llista de les ids de les Interseccions amb les que connecta. Seguidament es mostra un petit exemple de com queda un graf no dirigit representat per aquest diccionari.



ID	Nodes
1	2, 5
2	1, 3, 5
3	2, 4
4	3, 5, 6
5	1, 4, 2
6	4

Figura 39: Exemple de Graf

Aquesta classe només implementa els mètodes més utilitzats a l'hora de treballar amb grafs i no guarda cap tipus d'informació sobre les coordenades dels punts. Per utilitzar més funcions o les coordenades s'han de crear subclasses que ho permetin.

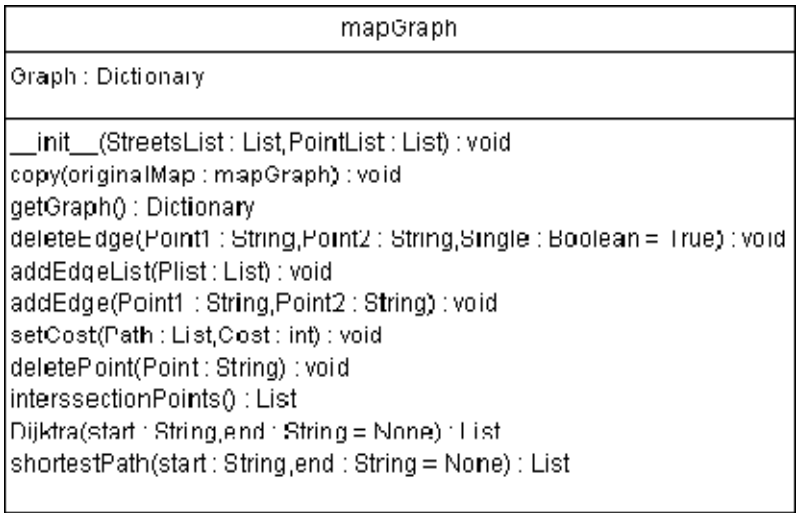


Figura 40: Classe mapGraph

- __init__(StreetsList, PointList)

  Aquest és el mètode constructor. Parteix d'una llista d'identificadors de nodes (*PointList*) i d'una altre llista amb una llista de camins (*StreetsList*) per especificar les relacions entre els nodes. Es genera un diccionari amb el format explicat anteriorment i s'assigna a l'atribut Graph de cada objecte d'aquesta classe.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció __init__(self, StreetsList, PointList):
 2      Graph= nou Diccionari amb prioritat
 3      per cada punt point de PointList fer:
 4          ID:= point.getAttribute('id')
 5          Graph[ID]:=Diccionari Buit
 6      per cada carrer street de StreetsList fer:
 7          StPoints:= Tots els elements del tipus nd de street
 8          sLenght:=nº d'elements de StreetPoints
 9          After:=StPoints[1]
10          Before:=Nul
11          i:=0
12          mentre i menor que sLenght fer:
13              current:=StreetPoints[i]
14              ref= current.getAttribute('ref')
15              edges=self.Graph[ref]
16              si After no és Nul llavors:
17                  Aref:= After.getAttribute('ref')
18                  edges[Aref]:=1
19              si Before no és Nul llavors:
20                  Bref:= Before.getAttribute('ref')
21                  edges[Bref]:=1
22              Before:=current
23              i:=i+1
24              si i+1 és més gran o igual a sLenght llavors:
25                    After:=Nul
26              altrament:
27                  After:=StreetPoints[i+1]
```

- copy(originalGraph)

  Copia l'atribut Graph de l'objecte *originalGraph* a l'atribut graph de l'objecte actual.

- getGraph()

  Retorna l'atribut Graph.

- deleteEdge(Point1, Point2, Single=True)

  Si *Single* és cert, s'elimina l'aresta que va de *Point1* a *Point2*. Si *Single* és fals, s'elimina l'aresta que va de *Point1* a *Point2* i la que va de *Point2* a *Point1*.

- addEdgeList(Plist)

  Donada una llista de punts (Plist) es creen les arestes necessàries per unir-los.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció addEdgeList(Plist):
2      FPoint:=Nul
3      Per cada punt point de Plist fer:
4          si FPoint és Nul llavors:
5              FPoint:=point
6          altrament:
7              addEdge(FPoint,point)
```

- addEdge(Point1, Point2,Single=False)

  Si Single és cert, afegeix una aresta entre *Point1* i *Point2.* Si és fals també crea l'aresta entre *Point2* i *Point1*.

- setCost(Path,Cost)

  Assigna un *Cost* a les arestes per les que passa el camí que formen els identificadors que hi ha a *Path.*

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció setCost(Path,Cost):
2      counter:=1
3      lenght:= nº d'elements de Path
4      mentre counter més petit que lenght fer:
5          first:= Path[counter-1]
6          second:= Path[counter]
7          Graph[first][second]:=Cost
8          Graph[second][first]:=Cost
9          counter:=counter+1
```

- deletePoint(Point)

  Elimina el punt (*Point*) i les arestes de les que formava part.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció deletePoint(Point):
2      pointList:=Graph[Point]
3      eliminar Graph[Point]
4      Per cada punt element de pointList fer:
5              eliminar Graph[element][Point]
```

- intersectionPoints()

  Retorna una llista amb l'identificador de tots els nodes intersecció (node amb més de dos arestes).

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció intersectionPoints():
2      List:=Llista Buida
3      Ids:=Obtenir totes les ids de Graph
4      Per cada punt point de ids fer:
5          nPunts:= nº d'elements dins la llista Graph[point]
6          si nPunts major que 2 llavors:
7              List.Afegeix(point)
8      retorna List
```

- Dijkstra(start,end=None)

  Aquest mètode implementa l'algorisme de Dijkstra. Aquest algorisme va ser concebut per l'holandès Edsger Dijkstra en 1959. Aquest algorisme troba la ruta amb el menor cost (per exemple, el camí més curt) entre un determinat vèrtex (node) del graf i qualsevol dels altres vèrtexs del graf.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció Dijkstra(Graf, origen):
2      per cada vèrtex v del Graf fer:
3          dist[v] := infinit
4          previ[v] := Nul
5      dist[origen] := 0
6      Q := Llistat de Tots els nodes
7      mentre Q no es buida:
8          u := vèrtex a Q amb menor dist[]
9          si dist[u] = infinit:
10             sortir
11         eliminar u de Q
12         per cada vei v de u
13             alt := dist[u] + dist_entre(u, v)
14             si alt < dist[v]:
15                 dist[v] := alt
16                 previ[v] := u
17     retorna previ
```

  En la versió que s'ha utilitzat en aquesta aplicació la llista de tots els nodes es troba ordenada per prioritat. Això s'ha fet perquè l'eficiència de l'algorisme millori deixant-la en $O((|E|+|V|) \log |V|)$ en comptes de $O(|V|^2+|E|) = O(|V|^2)$ de la versió original. També s'ha fet que juntament amb el previ (predessessors) retorni un llistat de les distàncies finals.
  Aquest algorisme es va implementar per ser utilitzat en una versió de l'algorisme detector d'illes, i tot i que en la versió final no s'utilitza, s'ha cregut que pot ser útil per a futures aplicacions d'aquesta classe.

- shortestPath(start,end=None)

  Aquest mètode crida el mètode Dijkstra, del resultat d'aquest n'extreu i ordena el camí mínim per anar del node end al node start i el retorna.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció shortestPath(self, start,end=None):
2      FinalDistances,Predecessors=Dijkstra(start,end)
3      Path:=Llista Buida
4      mentre cert fer:
5          Path.Afegir(end)
6          si end és igual a start llavors:
7               Sortir
8          si no existeix end dins de Predecessors llavors
9               Sortir
10         End:=Predecessors[end]
11     Path:=Girar l'orde de la llista Path
12     retorna Path
```

4.3.4 Classe osmGraph

Aquesta classe és una especialització de la classe anterior que permet treballar millor amb grafs provinents de fitxers OSM. Els objectes d'aquesta classe a part de tenir l'atribut Graph igual que la classe anterior també tenen dos atributs més, l'Streets i el Coordinates. Els dos són del tipus diccionari, en el primer es guarda la id del carrer com a índex i el llistat dels punts que formen aquest carrer com a dades. En el segon és guarda la id dels punts com a índex i les seves coordenades com a dades.

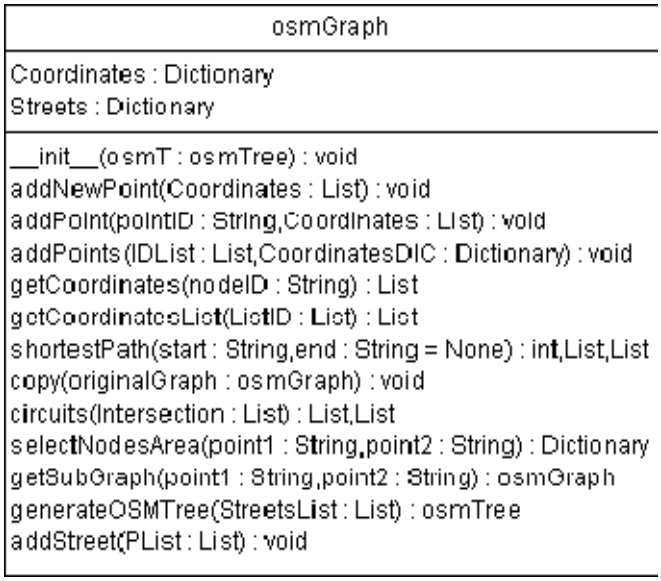


Figura 41: Classe osmGraph

- __init__(osmT)

  Aquest és el mètode constructor. A partir d'un objecte osmTree (o*smT*) genera el diccionari Graph utilitzant el resultat del mètode getStreetsList() de l'objecte osmT com a *StreetsList* i el mètode getNodeList() com a *PointList* (veure mètode __init__ de la classe anterior). Per omplir el diccionari Coordinates es fa un recorregut per tots els elements de tipus node de l'objecte osmT, guardant per a cada un d'ells, la seva id i les seves coordenades dins del

diccionari. Per omplir el diccionari Streets es fa un recorregut per tots els elements de tipus way de l'objecte osmT guardant per cada un d'ells la id del carrer i el llistat dels punts que el forment. Per més informació mirar els mètodes getStreetsList() i getNodeList() de la classe osmTree.

A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció __init__( osmT):
2
3      si osmT no és Nul:
4          Streets:= osmT.getStreetsList()
5          Nodes:= osmT.getNodeList()
6          Constructor del pare(Streets,Nodes)
7          Coordinates:=Streets:=Diccionari Buit
8          per cada carrer street de Streets fer:
9              streetPID:=Llista Buida
10             StreetPoints:=street.getElementsByTagName('nd')
11             per cada punt element de StreetPoints fer:
12                 ref:= element.getAttribute('ref')
13                 streetPID.Afegeix(ref)
14                 ID:= street.getAttribute('id')
15                 self.Streets[ID]:=streetPID
16
17         per cada punt point de Nodes fer:
18             nodeID=point.getAttribute('id')
19             Coordinates[nodeID]=osmT.getCoordinates(nodeID)
20     altrament:
21         Constructor del pare([],[])
22         self.Coordinates:=Streets:=Diccionari Buit
```

- addNewPoint(Coordinates)

  Genera una nova id de node mirant quina és la id actual més alta i sumant-li 1, i l'afegeix al diccionari Graph i al diccionari Coordinates, en aquest últim cas li assigna el paràmetre *Coordinates* com a dada. Retorna la id creada.

- addPoint(pointID, Coordinates)

  Afegeix la *pointID* als diccionaris Graph i Coordinates, en aquest últim cas li assigna el paràmetre *Coordinates* com a dada.

- addPoints(IDList,CoordinatesDIC)

  Igual que el mètode anterior però en aquest cas per un llistat de punts

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció addPoints(IDList,CoordinatesDIC):
2      lenght:=nº d'elements de la llista IDList
3      i:=0
4      mentre i menor que lenght fer:
5          addPoint(IDList[i],CoordinatesDIC[IDList[i]])
6          i=i+1
```

- addStreet(PList)

  Genera una nova id de carrer mirant quina és la id actual més alta i sumant-li 1, l'afegeix al diccionari Streets, i li assigna els punts de *PList* com a dada.

- getCoordinates(nodeID)

  Retorna les coordenades del punt (*nodeID*)

- getCoordinatesList(ListID)

  Retorna les coordenades dels punts de *ListID*

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció getCoordinatesList(ListID):
2      CList:=Llista Buida
3      per cada id element de ListID fer:
4          CList.Afegeix(getCoordinates(element))
5      retorna CList
```

- shortestPath(start,end=None)

  Igual al mètode amb el mateix nom de la classe anterior, però en aquest cas a part del camí mínim també retorna un llistat de les coordenades dels punts que formen aquest camí i el cost total d'aquest.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció shortestPath( start,end=None):
2      FinalDistances,Predecessors:=Dijkstra(start,end)
3      Path:=Coordinates:=Llista Buida
4      Distance:=-1
5      si existeix end dins de Predecessors llavors:
6          distance:=FinalDistances[end]
7      mentre cert:
8          Coordinates.Afegir(getCoordinates(end))
9          Path.Afegir(end)
10         si end és igual a start llavors:
11              Sortir
12         si no existeix end dins de Predecessors llavors
13              distance:=-1
14              Sortir
15         End:=Predecessors[end]
16     Path:=Girar l'orde de la llista Path
17     Coordinates:=Girar l'orde de la llista Coordinates
18     retorna distance, Path i Coordinates
```

- copy(originalGraph)

  Igual al mètode amb el mateix nom de la classe anterior però en aquest cas a part de copiar l'atribut Graph també copia els atributs Coordinates i Streets.

- circuits(Intersection)

  Donada una intersecció (*Intersection*) retorna una llista dels identificadors i una altre de les coordenades de tots els circuits tancats que passen per aquest punt. Perquè aquest mètode funcioni correctament les arestes han d'estar duplicades, veure figura 42. Aquesta classe per defecte ja les crea així, però si s'afegeixen arestes s'ha de recordar de no posar el paràmetre Single del mètode addEdge a cert. Per més informació anar a l'apartat 4.3.3.

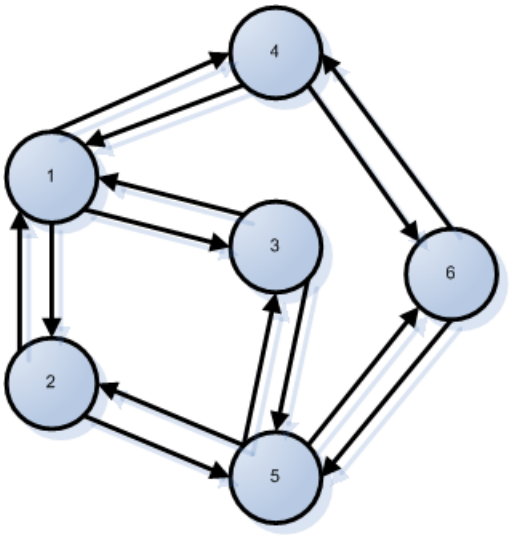


Figura 42: Exemple Graf amb arestes duplicades

Per detectar un circuit tancat es calculen tots els angles dels punts que tenen connexió amb l'actual i sempre s'escull com següent el que tingui un angle positiu menor. Això es repeteix fins que es tornar a arribar al node inicial *Intersection*, veure figura 43.

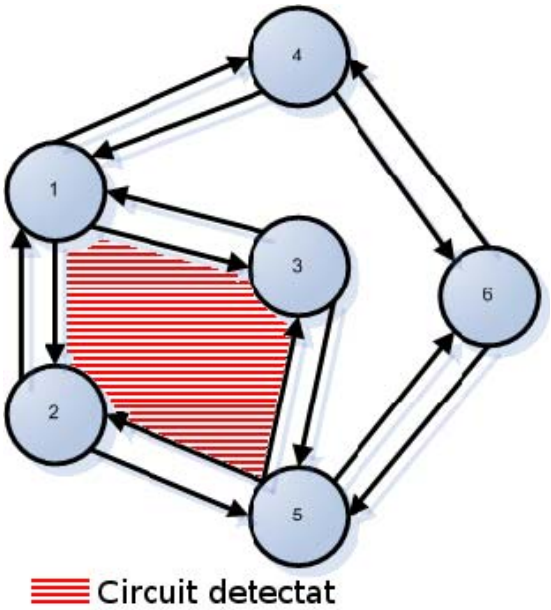


Figura 43: Exemple Circuit

Un cop s'ha aconseguit la illa, s'eliminen totes les arestes que en formen part, però no les inverses, veure figura 44.

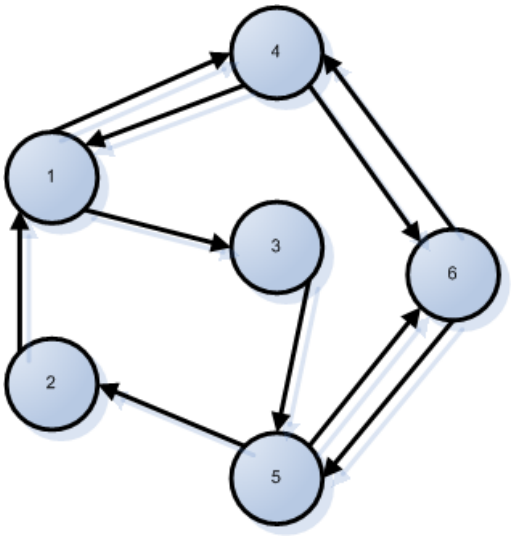


Figura 44: Graf amb les arestes corresponents al circuit detectat eliminades

Aquest mètode a part de totes les illes interiors també retorna una illa que conté tots els nodes dels límits del mapa, veure figura 45.

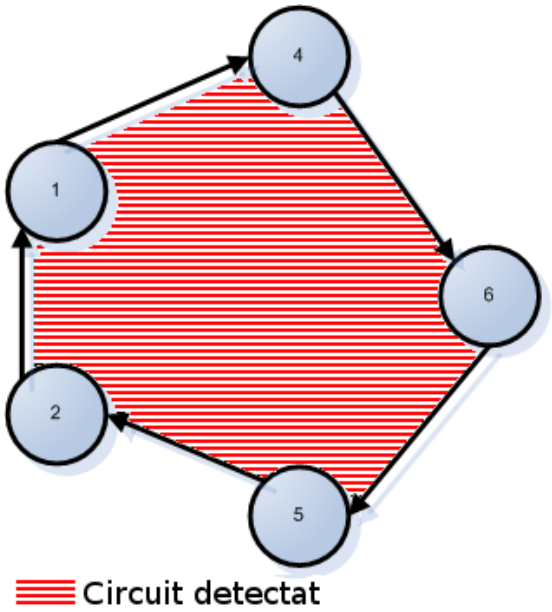


Figura 45: Illa que correspon a tot el contorn extern.

Seguidament es mostra l'algorisme en pseudocodi per aclarir-ne el funcionament.

```
 1  funció circuits(Intersection):
 2
 3      aTable:=blocksID:=blocksList:=Sedges:=llista buida
 4      Sedges:= Graph[Intersection]
 5      per cada aresta edge a Sedges fer:
 6          circuit:=llista amb només l'element Intersection
 7          Mpoint:=edge
 8          Spoint:=Intersection
 9          deleteEdge(Spoint,Mpoint)
10          points:=Graph[Mpoint]
11          mentre Mpoint diferent de Intersection fer:
12              si points és una llista buida llavors:
13                  Path:=llista buida
```

```
14                  Sortir del mentre
15
16              per cada punt point a points fer:
17                  CMpoint:= Coordinates[Mpoint]
18                  CSpoint:= Coordinates[Spoint]
19                  Cpoint:= Coordinates[point]
20                  angle=geo.angle(CMpoint,CSpoint,Cpoint)
21
22                  si angle es més gran que pi llavors:
23                      angle=angle-2*pi
24                  aTable.afegir(llista(angle,point))
25
26              Path.afegir (Mpoint)
27              Spoint:=Mpoint
28              Mpoint:=geo.minLeftangle(aTable).posició(1)
29              deleteEdge(Spoint,Mpoint)
30              aTable:= points:= llista buida
31              points.afegir(Graph[Mpoint])
32
33          si Path no és una llista buida llavors:
34              blocksID.afegir(Path)
35              blocksList.afegir(self.getCoordinatesList(Path))
36
37      retorna blocksID i blocksList
```

- selectNodesArea(point1,point2)

  Donats dos punts (*point1* i *point2*), retorna un diccionari d'interseccions amb l'identificador de cada intersecció com a índex i les coordenades com a dades de totes les interseccions tals que les seves coordenades es troben dins del rectangle definit per les coordenades dels dos punts donats.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció selectNodesArea(point1,point2):
 2      result:= Diccionari Buit
 3      lonP1:=point1[0]
 4      lonP2:=point2[0]
 5      latP1:=point1[1]
 6      latP2:=point2[1]
 7      points:=Totes les ids de Coordinates
 8      per cada punt point de points fer:
 9          Lon:=Coordinates[point][0]
10          Lat:=Coordinates[point][1]
11          coord=(Lon,Lat)
12          LonC1:= si Lon entre lonP1 i lonP2
13          LonC2:= si Lon entre lonP2 i lonP1
14          LatC1:= si Lat entre lonP1 i lonP2
15          LatC2:= si Lat entre latP2 i latP1
16          si (LonC1 o LonC2) i (LatC1 o LatC2) llavors:
17              result[point]:=Coordinates[point]
18      retorna result
```

- getSubGraph(point1,point2)

  Donats dos punts (*point1* i *point2*), retorna objecte osmGraph amb totes les interseccions, arestes i carrers tals que les seves coordenades es troben dins del rectangle definit per les coordenades dels dos punts donats.
  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció getSubGraph(point1,point2):
2      CoordinatesNodes:=selectNodesArea(point1,point2)
3      Nodes:=totes les ids de CoordinatesNodes
4      subGraph:=Nou osmGraph(Nul)
5      subGraph.addPoints(Nodes,CoordinatesNodes)
6      OriginalStreets:= totes les ids de Streets
7      per cada carrer OStreet de OriginalStreets fer:
8          OPoints:=Streets[OStreet]
9          NPoints:=Llista Buida
10         per cada punt point de OPoints fer:
11             si point existeix dins de Nodes fer:
12                 NPoints.Afegir(point)
13         lenght:=nº d'elements dins de NPoints
14         si lenght és major que 1 llavors:
15             subGraph.Streets[OStreet]:=NPoints
16             subGraph.setEndgesStreet(NPoints)
17
18     retorna subGraph
```

- GenerateOSMTree()

  Retorna un objecte osmTree a partir de les dades guardades als atributs Streets i Coordinates.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció GenerateOSMTree():
2      cityOSM:= Nou osmTree(Nul)
3      cityOSM.addNodes(Coordinates)
4      keys=Obtenir tots els ID de Streets
5      per cada carrer street de keys fer:
6          cityOSM.addStreet(Streets[street],street)

7      cityOSM.setBounds()
8      retorna cityOSM
```

4.3.5 Classe osmStadistics

Aquesta classe calcula totes les estadístiques que s'utilitzen a l'hora de generar la xarxa de carrers. També en calcula el nivell i classifica les interseccions segons aquest.

El carrers és classifiquen utilitzant la següent taula de correlació entre el tipus de via del mapa OSM i el nivell assignat, guardada com a l'atribut de classe __streetHierarchy. Per a més informació sobre el tipus de vies accedir a l'annex1.

```
__streetHierarchy={   "motorway":9,
                      "motorway_link":9,
                      "trunk":8,
                      "trunk_link":8,
                      "primary":7,
                      "primary_link":7,
                      "secondary":6,
                      "secondary_link":6,
                      "tertiary":5,
                      "unclassified":4,
                      "road":3,
                      "residential":2,
                      "living_street":1,
                      "service":1,
                    }
```

Figura 46: Taula __streetHierarchy

Cadascun dels objectes d'aquesta classe guarda un diccionari d'interseccions on es guarden la id de cada intersecció com a índexs i un llistat de les estadístiques del punt en aquest ordre : nivell, tortuositat, mitja de la distància, variança de la distància, mitja de l'angle i la variança de l'angle. També guarden un altre diccionari anomenat Level on l'índex indica el nivell i les dades són un llistat de totes les interseccions d'aquest nivell.

A l'hora de calcular les totes estadístiques, menys el nivell, s'utilitzen els mètodes distància i angle de la classe geo, depenent del que es necessiti.

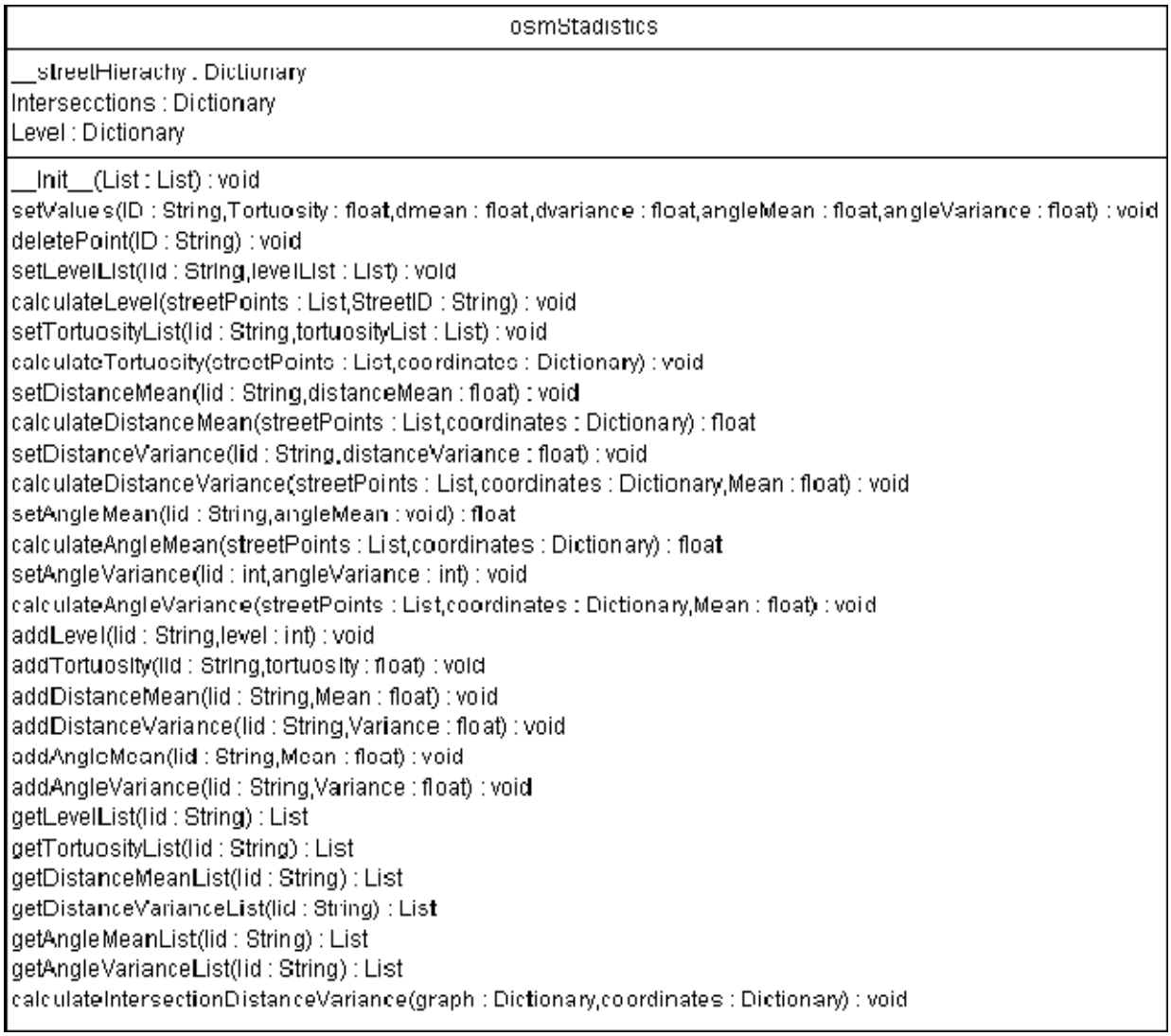


Figura 47: Classe osmTree

- __init__(List):

  Aquest és el mètode constructor. A partir d'una llista d'interseccions afegeix aquestes al diccionari intersections de cada objecte com a índex i afageix una llista buida a cadascuna de les estadístiques del camp de dades.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció __init__(List):
2      Intersections:=Level:=Diccionari Buit
3      Level:= Tortuosity:=dmean:=dVariance:=LlistaBuida
4      anglemean:=angleVariance:=LlistaBuida
5      V:=(Level,Tortuosity,dmean,dVariance, anglemean)
6      V.Afegeix(angleVariance)
7      per cada intersecció element de List fer:
8          self.Intersections[element]=V
```

- setValues(ID,tortuosity,dmean,dVariance,anglemean,angleVariance)

  Assigna els valors de tortuositat (*tortuosity*), mitja de la distància (*dmean*), la variança de la distància (*dVariance*), la mitja de l'angle (*anglemean*), i la variança de l'angle (*angleVariance*) al punt *ID*.

- deletePoint(ID)

  Elimina el punt amb identificador igual a *ID* del diccionari Intersections i busca a quin nivell/s es troba i també l'elimina de la llista d'aquests dins el diccionari Level.

- setLevelList(lid, levelList)

  Assigna al punt *lid* els nivells que hi ha a *levelList*.

- calculateLevel(streetPoints, StreetID)

  Aquest mètode assigna un nivell a carrer StreetID utilitzant la relació entre el tipus de carrer i el nivell que es troba a la taula __streetHierarchy, i l'afegeix utilitzant el mètode addLevel d'aquesta classe a cadascun dels punts que el formen.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció calculateLevel(streetPoints):
 2      Intersections:=Llista Buida
 3      Level:=0
 4      LTable:=__streetHierarchy
 5      per cada punt point de streetPoints fer:
 6          si point.nodeType és diferent a 3 fer:
 7              nodeID:= point.getAttribute('ref')
 8              si nodeID existeix a Intersections llavors:
 9                  intersections.append(nodeID)
10              altresi nodeID és Nul llavors:
11                  k:= point.getAttribute('k')
12                  si k és igual a highway fer:
13                      levelStr:= point.getAttribute('v')
14                      si levelStr existeix a LTable llavors:
15                          level:= LTable[levelStr]
16       per cada element de intersections fer:
17          addLevel(element,level)
```

- setTortuosityList(lid, tortuosityList)

  Assigna al punt *lid* les tortuositats que hi ha a *tortuositylList*.

- calculateTortuosity(streetPoints,coordinates)

  Calcula la tortuositat del carrer format pels punts que hi ha a *streetPoints* amb coordenades dins de *coordinates* i l'afegeix a cada punt del carrer utilitzant el mètode addTortuosity d'aquesta mateixa classe.

  La formula de la tortuositat és (D1+D2)/D3 on D1 i D2 representen la llarga de dos segments i D3 la distància en línia recte entre els punts inicial d'un d'aquests segments i el final de l'altre.

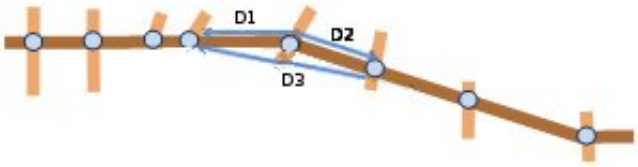


Figura 48: Distancies per calcular la tortuositat

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció calculateTortuosity(streetPoints,coordinates):
 2
 3      lastInt:=firstInt:=lastNode:=Nul
 4      intDistance:=tortuos:=nSegments:=0
 5      Tint:=Llista Buida
 6      Int:=Intersections
 7      gD:= geo.distance
 8      per cada punt point de streetPoints fer:
 9          si point.nodeType és diferent a 3 fer:
10              nodeID:= point.getAttribute('ref')
11              si nodeID existeix a coordinates llavors:
12                  si lastInt és Nul llavors:
13                     firstInt:=lastNode:=nodeID
14                     lastInt:=nodeID
15
16                  altresi nodeID és diferent a lastInt:
17                      nodeCoordinates:=coordinates[nodeID]
18                      lastCoordinates:=coordinates[lastNode]
19                      lCLon:= lastCoordinates[0]
20                      lCLat:= lastCoordinates[1]
21                      NLon:= nodeCoordinates[0]
22                      NLat:= nodeCoordinates[1]
23                      distance:= gD(lCLon,lCLat, NLon,NLat)
24                      lastNode:=nodeID
25                      intDistance:=intDistance+distance
26                      si nodeID existeix a Int llavors:
27                          Tint.Afegeix(nodeID)
28                          nodeCoordinates:=coordinates[nodeID]
29                          lastCoordinates=coordinates[lastInt]
30                          lCLon:= lastCoordinates[0]
31                          lCLat:= lastCoordinates[1]
32                          DistI:= gD(lCLon,lCLat, NLon,NLat)
33                          tortuos=tortuos+intDistance/DistI
34                          intDistance:=0
35                          lastInt:=nodeID
36                          nSegments:= nSegments+1
37      Tint:=Ordenar Tint
38      si nSegments és diferent de 0 fer:
39          tortuos=tortuos/nSegments
40      altrament:
41          tortuosity:=0
42      per cada punt element de intersections fer:
43          addTortuosity(element,tortuosity)
```

- setDistanceMean(Iid, distanceMean)

  Assigna al punt *Iid* les mitjanes de distància que hi ha a *distanceMean*.

- calculateDistanceMean(streetPoints,coordinates)

  Calcula la distància mitja dels segments que formen el carrer amb punts *streetPoints* i coordenades *coordinates*, l'afegeix a cada punt del carrer utilitzant el mètode addDistanceMean d'aquesta classe i finalment la retorna.

  La formula de la mitjana és $\overline{X} = \sum x / N$ on x és la longitud de cadascun dels segments del carrer actual i N és el nombre de segments que formen el carrer.

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1  funció calculateDistanceMean(streetPoints,coordinates):
 2      streetInts:= nodeCoords:= previousCoords:=Llista Buida
 3      distance:=SegmentsLenght:=Mean:=0
 4      previousNode:=FIntersection:=Nul
 5      per cada punt point de streetPoints fer:
 6          si point.nodeType és diferent a 3 fer:
 7              nodeID:= point.getAttribute('ref')
 8              Ei:= nodeID existeix a Intersections
 9
10              si Ei i previousCoords és Buida llavors:
11                  FIntersection:=nodeID
12                  previousCoords=coordinates[nodeID]
13
14              altresi Ei i nodeCoords no és Buida llavors:
15                  streetIntersections.Afegir(nodeID)
16                  nodeCoords:=coordinates[nodeID]
17                  pCLon:= previousCoords[0]
18                  pCLat:= previousCoords[1]
19                  NLon:= nodeCoords[0]
20                  NLat:= nodeCoords[1]
21                  dist:= dist+gD(lCLon,lCLat, NLon,NLat)
22                  previousCoords:=coordinates[nodeID]
23                  SegmentsLenght:= SegmentsLenght+dist
24                  dist:=0
25
26              altresi previousCoords és Buida i nodeID és Nul:
27                  nodeCoords:=coordinates[nodeID]
28                  pCLon:= previousCoords[0]
29                  pCLat:= previousCoords[1]
30                  NLon:= nodeCoords[0]
31                  NLat:= nodeCoords[1]
32                  dist:= dist+gD(lCLon,lCLat, NLon,NLat)
33                  previousCoordinates:=coordinates[nodeID]
34
35      length:= nº d'elements de streetIntersections
36      si length no és 0:
37          Mean:=SegmentsLenght/length
38          streetInts.Afegir(FIntersection)
39
40      per cada punt element de streetInts fer:
41          addDistanceMean(element,Mean)
42      retorna Mean
```

- setDistanceVariance(Iid, distanceVariance)

  Assigna al punt *Iid* les variàncies de distància que hi ha a *distanceVariance*.

- calculateDistanceVariance(streetPoints,coordinates,Mean)

  Calcula la variança de la distància dels segments que formen el carrer amb punts *streetPoints* i coordenades *coordinates* respecta a la mitja de la distància *Mean*. Afegeix la mitjana de totes les variances dels segments que formen el carrer a cada punt d'aquest utilitzant el mètode addDistanceVariance d'aquesta classe.

La formula de la variança és $\sigma^2 = (X - \mu)^2$ on X és la distància del segment actual i μ és la distància mitja del carrer.

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció calculateDistanceVariance(streetPoints,coords,Mean):
 2      streetInts:=nodeCoords:=previousCoords:= Llista Buida
 3      variance:=distance:=0
 4      previousNode:=FIntersection:=Nul
 5      per cada punt point de streetPoints fer:
 6          si point.nodeType és diferent a 3 fer:
 7              nodeID:= point.getAttribute('ref')
 8              Ei:= nodeID existeix a Intersections
 9              si Ei i previousCoords és Buida llavors:
10                  FIntersection:=nodeID
11                  previousCoords=coordinates[nodeID]
12
13              altresi Ei i nodeCoords no és Buida llavors:
14                  streetIntersections.Afegir(nodeID)
15                  nodeCoords:=coordinates[nodeID]
16                  pCLon:= previousCoords[0]
17                  pCLat:= previousCoords[1]
18                  NLon:= nodeCoords[0]
19                  NLat:= nodeCoords[1]
20                  dist:= dist+gD(lCLon,lCLat, NLon,NLat)
21                  previousCoords:=coordinates[nodeID]
22                  variance:=variance+(distance-Mean)^2
23                  dist:=0
24
25              altresi previousCoords és Buida i nodeID és Nul:
26                  nodeCoords:=coordinates[nodeID]
27                  pCLon:= previousCoords[0]
28                  pCLat:= previousCoords[1]
29                  NLon:= nodeCoords[0]
30                  NLat:= nodeCoords[1]
31                  dist:= dist+gD(lCLon,lCLat, NLon,NLat)
32                  previousCoordinates:=coordinates[nodeID]
33
34      length:= nº d'elements de streetIntersections
35
36      si length no és 0:
37          variance=variance/lenght
38          streetInts.Afegir(FIntersection)
39
40      per cada punt element de streetInts fer:
41          addDistanceVariance(element,Mean)
```

Aquest mètode finalment no s'ha utilitzat en aquesta aplicació, s'ha subtituit pel mètode calculateIntersectionDistanceVariance, ja que si es calculava la variança de la distància d'aquesta manera pot succeir que la probabilitat de qualsevol intersecció per connectar amb una altre sempre sigui 0. Per més informació mirar el mètode choosePoint a l'apartat 4.3.11. Això es produeix quan, per exemple per una intersecció, passen dos carrers molt constants en la longitud dels seus segments. Això significa que la variança de la distància d'aquests dos carrers és aproximadament 0. Al calcular la mitjana de totes les estadístiques de cadascuna de les interseccions, la variança de la distància continua siguent semblant a 0 però la mitja de la distància ha canviat (a no ser que la mitjana de les distàncies dels dos carrers fossin iguales), això comporta

que al calcular les probabilitats només siguin vàlids punts que es troben exactament a una distància igual a la mitja de la distància i això és molt improbable. Bàsicament el que passa és que la funció Gaussiana per calcular la probabilitat en aquest punt té una amplitud tant petita que tots els punts surten fora d'ella.

- calculateIntersectionDistanceVariance(graph,coordinates)

  Per cada Intersecció calcula la variança de la distància dels segments que passen per ella respecte al promitg de les mitjanes de les distàncies entre els segments de cadascun dels carrers que passen per ella. Veure figura 49.
  El paràmetre *graph* és un diccionari amb el mateix format que l'atribut Graph de la classe mapGraph, veure apartat 4.3.3. El paràmetre *coordinates* és un diccionari amb les coordenades de tots els punts de *graph* amb el mateix format que l'atribut Coordinates de la classe osmGraph, veure apartat 4.3.4

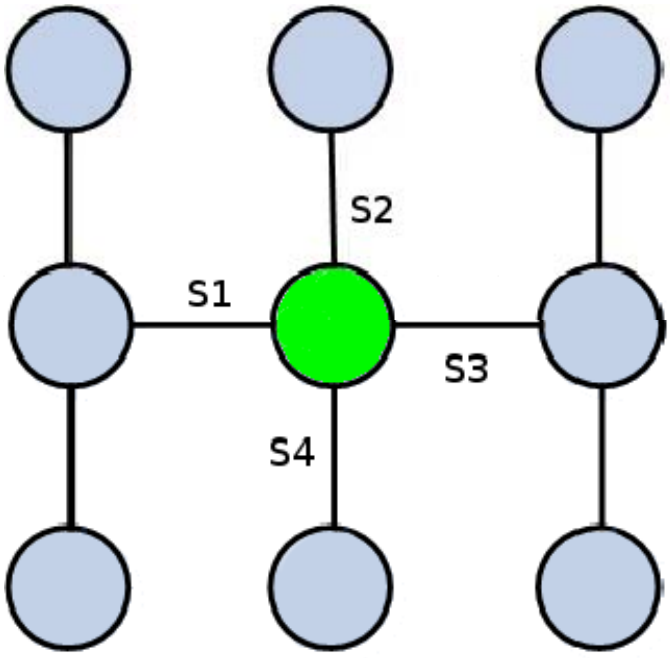


Figura 49: Classe osmTree

En el cas de la figura 49 la intersecció en verd, aquest mètode calcularia la variança de la següent forma:

MitjaCarrers= (longitud segment mitja carrer que conté S1 i S3) + (longitud segment mitja carrer que conté S2 i S4)/2

VariançaS1= $(LongitudS1- MitjaCarrers)^2$

VariançaS2= $(LongitudS2- MitjaCarrers)^2$

Les VariançaS3 i VariançaS4 es calculen igual que les anteriors.

VariançaIntersecció=(S1+S2+S3+S4)/4

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció calculateIntersectionDistanceVariance(graph,coords):
 2      Keys:= Llista de ids de Intersections
 3      per cada clau key de keys fer:
 4          DM:=getDistanceMean(key)
 5          si DM és Nul llavors:
 6              setDistanceVariance(Intersections[key],0)
 7          altrament:
 8              MDM=Suma de tots els elements de DM
 9              length:=nº d'elements de DM
10              MDM:=MDM/lenght
11              Fpoints:=graph[key]
12              CpointCoords:=coordinates[key]
13              Variance:=0
14              per cada punt point de Fpoints fer:
15                  pointCoords:=coordinates[point]
16                  FLon:= CpointCoords[0]
17                  FLat:= CpointCoords[1]
18                  NLon:= pointCoords [0]
19                  NLat:= pointCoords [1]
20                  dist:= dist+gD(NLon,NLat,FLon,FLat)
21                  variance:= variance+(distance-MDM)²
22
23              lenghtF:=nº d'elements de Fpoints
24              Mvar:= variance/lenghtF
25              setDistanceVariance(Intersections[key],Mvar)
```

- setAngleMean(Iid, angleMean)

  Assigna al punt *Iid* les mitjanes de l'angle que hi ha a angl*eMean*

- calculateAngleMean(streetPoints,coordinates)

  Calcula la mitjana de l'angle, que formen dos segments consecutius units per un punt, per cadascun dels punts que formen el carrer amb punts *streetPoints* i coordenades *coordinates*. Un cop calculada, s'afegeix a cada punt del carrer utilitzant el mètode addAngleMean d'aquesta classe i també es retorna.

  La formula de la mitjana és $\overline{X} = \sum x / N$ on x és l'angle que hi ha entre dos segments consecutius del carrer actual i N és el nombre de punts que formen el carrer.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció calculateAngleMean(streetPoints,coordinates):
 2      streetInts:=SNode:=MNode:=Llista Buida
 3      angle:=Mean:=nAngles:=0
 4      FIntersection:=Nul
 5      per cada punt point de streetPoints fer:
 6          si point.nodeType és diferent a 3 fer:
 7             nodeID:= point.getAttribute('ref')
 8             Ei:= nodeID existeix a Intersections
 9             Empty:=SNode i MNode són Buides
10             si Ei i SNode és Buida llavors:
11                  streetIntersections.Afegir(nodeID)
12                  SNode:=coordinates[nodeID]
13
14             altresi Ei i MNode no és Buida llavors:
15                  nAngles:= nAngles+1
```

```
16                  streetIntersections.Afegir(nodeID)
17                  FNode:=coordinates[nodeID]
18                  Angle:= angle+geo.angle(MNode,SNode,FNode)
19                  SNode:=MNode
20                  MNode:=FNode
21
22              Altresi Empty i nodeID és Nul llavors:
23                  MNode:=coordinates[nodeID]
24
25              altresi SNode és Buida i nodeID és Nul fer:
26                  nAngles:= nAngles+1
27                  FNode:=coordinates[nodeID]
28                  Angle:=angle+geo.angle(MNode,SNode,FNode)
29                  SNode:=MNode
30                  MNode:=FNode
31
32      si nAngles és 0 fer:
33          Mean:=angle/nAngles
34          si Mean és inferior a 0 fer:
35              Mean:=2*math.pi+Mean
36
37      per cada punt element de streetInts fer:
37          addAngleMean(element,Mean)
39      retorna Mean
```

- setAngleVariance(Iid, angleVariance)

  Assigna al punt *Iid* les variàncies de l'angle que hi ha a angle*Variance*.

- calculateAngleVariance(streetPoints,coordinates,Mean)

  Calcula la variança de l'angle dels segments que formen el carrer amb punts *streetPoints* i coordenades *coordinates* respecte a la mitja de l'angle *Mean*. Afegeix la mitjana de totes les variances dels segments que formen el carrer a cada punt d'aquest utilitzant el mètode addDistanceVariance d'aquesta classe.

  La formula de la variança és $\sigma^2 = (X - \mu)^2$ on X és la distància del segment actual i μ és la distància mitja del carrer.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció calculateAngleVariance(streetPoints,coords,Mean):
 2       streetInts:=SNode:=MNode:=Llista Buida
 3      angle:=Mean:=nAngles:=variance:=0
 4      FIntersection:=Nul
 5      per cada punt point de streetPoints fer:
 6          si point.nodeType és diferent a 3 fer:
 7             nodeID:= point.getAttribute('ref')
 8             Ei:= nodeID existeix a Intersections
 9             Empty:=SNode i MNode són Buides
10             si Ei i SNode és Buida llavors:
11                  streetIntersections.Afegeix(nodeID)
12                  SNode:=coordinates[nodeID]
13
14              altresi Ei i MNode no és Buida llavors:
15                  nAngles:= nAngles+1
16                  streetIntersections.Afegir(nodeID)
17                  FNode:=coordinates[nodeID]
```

```
18                  angle:= geo.angle(MNode,SNode,FNode)
19                  si angle és inferior a 0 fer:
20                      angle:=2*math.pi+Mean
21                  variance:= variance+ (angle-Mean)²
22                  SNode:=MNode
23                  MNode:=FNode
24
25              Altresi Empty i nodeID és Nul llavors:
26                  MNode:=coordinates[nodeID]
27
28              altresi SNode és Buida i nodeID és Nul fer:
29                  nAngles:= nAngles+1
30                  FNode:=coordinates[nodeID]
31                  angle:=geo.angle(MNode,SNode,FNode)
32                  si angle és inferior a 0 fer:
33                      angle:=2*math.pi+Mean
34
35                  variance= variance+ (angle-Mean)²
36                  SNode:=MNode
37                  MNode:=FNode
38
39      si nAngles és 0 fer:
40          variance:=variance/nAngles
41
42      per cada punt element de streetInts fer:
43          self.addAngleVariance(element,variance)
```

- addLevel(Iid, level)

  Afegeix el nivell *level* a la llista que es troba a posició Level del punt *Iid* dins el diccionari Intersections. També agrega a aquest punt al lloc corresponent a *level* dins del diccionari Level.

- addTortuosity(Iid, tortuosity)

  Afegeix la tortuositat *tortuosity* a la llista que es troba a posició tortuosity del punt *Iid* dins el diccionari Intersections.

- addDistanceMean(Iid, Mean)

  Afegeix la mitja de la distància *Mean* a la llista que es troba a posició DistanceMean del punt *Iid* dins el diccionari Intersections.

- addDistanceVariance(Iid, Variance)

  Afegeix el la variança de la distància *Variance* a la llista que es troba a posició DistanceVariance del punt *Iid* dins el diccionari Intersections.

- addAngleMean(Iid, Mean)

  Afegeix la mitja de l'angle *Mean* a la llista que es troba a posició AngleMean del punt *Iid* dins el diccionari Intersections.

- addAngleVariance(Iid, Variance)

  Afegeix el la variança de l'angle *Variance* a la llista que es troba a posició AngleVariance del punt *Iid* dins el diccionari Intersections.

- getLevelList(Iid)

  Retorna la llista de nivells del punt *Iid*.

- getTortuosityList(Iid)

  Retorna la llista de tortuositats del punt *Iid*.

- getDistanceMeanList(Iid)

  Retorna la llista de les mitjanes de distància del punt *Iid*.

- getDistanceVarianceList(Iid)

  Retorna la llista de les variàncies de la distància del punt *Iid*.

- getAngleMeanList(Iid)

  Retorna la llista de les mitjanes de l'angle del punt *Iid*.

- getAngleVarianceList(Iid)

  Retorna la llista de les variàncies de l'angle del punt *Iid*.

4.3.6 Classe dataHandler

Aquesta classe és el nexe d'unió entre la resta de classes importants d'aquest projecte. S'ocupa de fer operacions especifiques per aquest projecte.

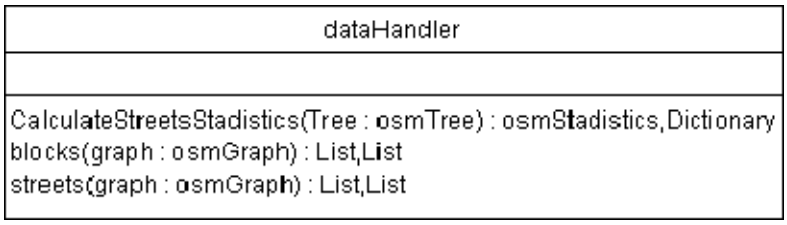


Figura 50: Classe dataHandler

- CalculateStreetStatistics(Tree)

  Donat un objecte de la classe osmTree (*Tree*), crea un objectes de la classe osmStatistics i en calcula:

  - el nivell de tots els carrers (veure mètode calculateLevel de la classe anterior).
  - la tortuositat de tots els carrers (veure mètode calculateTortuosity de la classe anterior)
  - la mitjana de la longitud dels segments de cada carrer (veure mètode calculateDistanceMean de la classe anterior)
  - la mitjana de l'angle dels segments que formen cadascun dels carrers (veure mètode calculateAngleMean de la classe anterior)

- la variança de l'angle dels segments que formen cadascun dels carrers, respecte a la mitjana de l'angle calculada anteriorment. (veure mètode calculateAngleVariance de la classe anterior)

- la variança de la longitud dels segments que passen per una Intersecció respecte a la mitjana de la longitud dels carrers que passen per ella. (veure mètode calculateIntersectionDistanceVariance de la classe anterior).

Un cop s'ha finalitzat el càlcul de totes les estadístiques, es fa un recorregut per cadascuna de les interseccions de l'objecte osmStatistics per calcular la mitjana de cadascuna de les estadístiques d'aquestes interseccions (menys pel nivell i la variança de la longitud) i substituir la llista de valors que tenen assignada per la mitja d'aquestes llistes que s'acaba de calcular. Finalment retorna l'objecte osmStatistics resultant.

Seguidament es mostra el pseudocodi d'aquest mètode.

```
 1  funció CalculateStreetStadistics(Tree):
 2     streets:=Tree.getStreetsList()
 3      Nodes:=Tree.getIntersectionNodes()
 4      std:=osmStadistics(Nodes)
 5      graph:=mapGraph(streets,Tree.getNodeList())
 6      DicG:= graph.getGraph()
 7      coord:=Tree.getPointsCoordinates()
 8      per cada carrer street de streets fer:
 9          ID:=obtenir id del carrer street
10          elmts:=obtenir id punts del carrer street
11          std.calculateLevel(elmts,ID)
12          std.calculateTortuosity(elmts,coord,ID)
13          std.calculateDistanceMean(elmts,coord,ID)
14          AM:=std.calculateAngleMean(elmts,coord,ID)
15          std.calculateAngleVariance(elmts,coord,AM,ID)
16
17      stad.calculateIntersectionDistanceVariance(DicG,coord)
18      per cada intersecció intr de Nodes fer:
19          T=calcular mitja std.getTortuosityList(intr)
20          DM=calcular mitja std.getDistanceMeanList(intr)
21          DV= std.getDistanceVarianceList(intr)
22          AM= calcular mitja std.getAngleMeanList(intr)
23          AV= calcular mitja std.getAngleVarianceList(intr)
24          std.setValues(intr,T,DM,DV,AM,AV)
25
26     retorna std i coord
```

- blocks(graph)

  Retorna les coordenades dels punts que formen les illes que hi ha a l'objecte de la classe osmGraph *graph*.

  Per buscar les illes es crida el mètode circuits de l'objecte *graph* per a cadascuna de les interseccions que conté. Un cop s'han trobat totes, s'elimina la més exterior, ja que el mètode utilitzat per trobar-les retorna una macro illa que conté tots els nodes dels límits del mapa (veure mètode circuits de la classe osmGraph i la figura 45). Un cop eliminada aquesta illa, es retorna la llista dels identificadors dels punts que formen cada illa i una altre llista igual que l'anterior però en comptes dels identificadors, retorna les coordenades de

cada punt. Aquest últim és una llista de conveniència per treballar amb altres aplicacions com ara l'urbanEngine.

Seguidament es mostra el pseudocodi d'aquest mètode.

```
 1  funció blocks(graph):
 2      FinalblocksList:=FinalblocksID:=blocksList:=Llista Buida
 3      blocksID:=Llista Buida
 4      intersections:=graph.intersectionPoints()
 5
 6      per cada intersecció intersection de intersections fer:
 7          blocksID,blocksList=graph.circuits(intersection)
 8          per cada illa block de blocksList fer:
 9              FinalblocksList.Afegir(block)
10          per cada id blockID de blocksID fer:
11              FinalblocksID.Afegir(blockID)
12
13          blocksList:=Llista Buida
14      maxLen:=0
15      lenght:=nº d'elements de FinalblocksID
16      i:=0
17      mentre i menor que lenght fer:
18          L1:= nº d'elements de FinalblocksID[i]
19          L2:= nº d'elements de FinalblocksID[maxLen]
20          si L1 major que L2 llavors:
21              maxLen:=i
22          i:=i+1
23      eliminar FinalblocksID[maxLen]
24      eliminar FinalblocksList[maxLen]
25      retorna FinalblocksList i FinalblocksID
```

- streets(graph)

  Retorna una llista dels identificadors dels punts que formen cada carrer i una altre llista igual que l'anterior però, en comptes dels identificadors, retorna les coordenades de cada punt.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció streets(graph):
 2      Keys:=ids de graph.Streets
 3      StreetsList:=StreetsPList:=Llista Buida
 4      Per cada id key de keys fer:
 5          PList:=graph.Streets[key]
 6          StreetsList.Afegir(PList)
 7          StreetsPList.Afegir(graph.getCoordinatesList(PList))
 8      retorna StreetsPList,StreetsList
```

4.3.7 Classe Configuration

Aquesta classe simplement guarda les variables de configuració que l'usuari pot modificar per generar xarxes de carrers al seu gust.

La funció de cadascuna d'aquestes variables serà explicada en el apartat 4.3.11 en el mètode randomWalk.

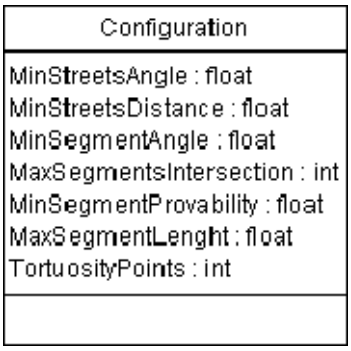


Figura 51: Classe Configuratiom

4.3.8 Classe CDF

Aquesta classe calcula la "Comulative Distribution Funcion" (explicada a l'apartat 3.2.1.2)

Figura 52: Classe CDF

- cdf(PList)

  Aquest mètode rep una llista(PList) en format PList [[Probabilitat,ID],...] i seguint la "Comulative Distribution Funcion" (explicada a l'apartat 3.2.1.2), retorna un ID.

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció cdf(PList):
 2      Point:=Nul
 3      si PList no és Nul llavors:
 4          PList:= PList ordenada de major a menor
 5          Aux:=0
 6          posList:=llista de 0 a nº elements de PList
 7          per cada posicio pos de posList fer:
 8              aux:=aux+PList[pos][0]
 9              PList[pos][0]:=aux
10          Lenght:=nº d'elements de PList
11          Rand:= Generar nº aleatori entre 0 i 1
12          Rand:=Rand* PList[(lenght-1)][0]
13          Point:=PList[0][1]
14          Per cada nombre pos2 de posList fer:
15              si rand és més gran que PList[pos2][0] llavors:
16                  Point:=PList[pos2][1]
17      retorna Point
```

4.3.9 Classe geo

Aquesta classe agrupa tots els mètodes que treballen amb dades geomètriques i geogràfiques.

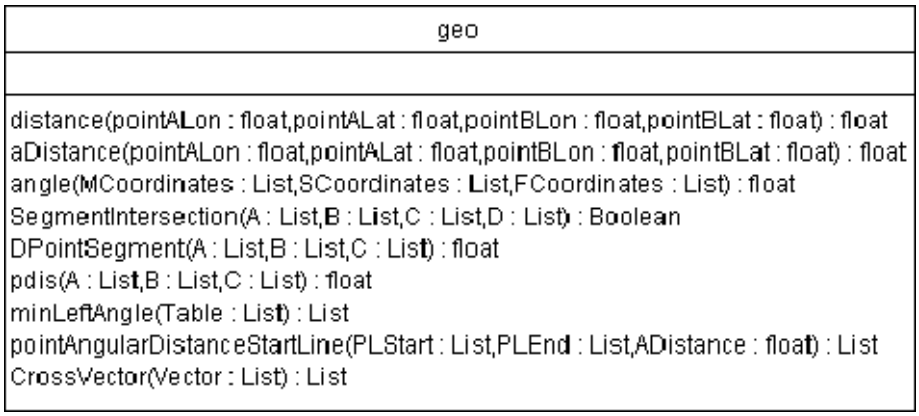


Figura 53: Classe geo

- distance(pointALon,pointALat,pointBLon,pointBLat)

  Retorna la distància en kilòmetres entre el punt A (*pointALon*,*pointALat*) i el punt B(*pointBLon*,*pointBLat*), aquests punts es troben expressats en format de longitud i latitud.

  Per calcular aquesta distància s'utilitza la formula de Haversine, s'ha optat per aquesta en comptes de les altres possibles ja que dona una precisió superior a la majoria i en els casos en que la dona inferior la complexitat computacional dels altres mètodes és molt superior. A continuació és mostra la formula utilitzada.

  R = Radi de la terra (Radi mitjà = 6371 km)
  $\Delta lat = lat_2 - lat_1$
  $\Delta long = long_2 - long_1$
  $a = \sin^2(\Delta lat/2) + \cos(lat_1).\cos(lat_2).\sin^2(\Delta long/2)$
  $c = 2.\text{atan2}(\sqrt{a}\ ,\ \sqrt{(1-a)}\ )$
  $d = R.c$

- aDistance(pointALon,pointALat,pointBLon,pointBLat)

  Retorna la distància angular en radiants entre el punt A (*pointALon*,*pointALat*) i el punt B(*pointBLon*,*pointBLat*), aquests punts es troben expressats en format de longitud i latitud.

  Per calcular aquesta distància es calcula el mòdul ($\overline{V}$) del vector que va del punt A al B. $\overline{V} = \sqrt{Lat^2 + Lon^2}$ on $Lat = LatitudPuntB - LatitudPuntA$ i $Lon = LongitudPuntB - LongitudPuntA$

- angle(MCoordinates,SCoordinates,FCoordinates)

  Retorna l'angle entre els vectors *SCoordinates-MCoordinates* i *MCoordinates-FCoordinates*.

Per calcular aquest angle es calculen els vectors que van de *SCoordinates* a *MCoordinates* (V1) i el que va de *MCoordinates* a *FCoordinates*(V2) i s'aplica la fòrmula següent: α=atan2(v2.lay,v2.lon) - atan2(v1.lat,v1.lon).

- SegmentIntersection(A,B,C,D)

  Retorna un Booleà que indica si el vector *AB* i el vector *CD* intersequen.

  Aquest booleà es calcula comparant si els dos resultats de les següents equacions es troben entre 0 i 1. En cas afirmatiu, vol dir que els dos segments es creuen.

$$u_a = \frac{(x4 - x3) * (y1 - y3) - (y4 - y3) * (x1 - x3)}{(y4 - y3) * (x2 - x1) - (x4 - x3) * (y2 - y1)}$$

$$u_b = \frac{(x2 - x1) * (y1 - y3) - (y2 - y1) * (x1 - x3)}{(y4 - y3) * (x2 - x1) - (x4 - x3) * (y2 - y1)}$$

  On x1=longitud del punt A, x2=longitud del punt B, x3=longitud del punt C, x4=longitud de D, y1=latitud del punt A, y2= latitud del punt B, y3= latitud del punt C i y4=latitud de D

- DPointSegment(A,B,C)

  Retorna la distància entre el vector *AB* i el punt *C*.

  Aquesta distància es calcula depenent del valor de la seguent equació.

$$u = \frac{(x3 - x1) * (x2 - x1) + (y3 - y1) * (y2 - y1)}{\|p2 - p1\|^2}$$

  Segons el valor del paràmetre u, es prenen diferents accions:

  - u menor a 0

    La distància és igual a la distància entre el punt C i el punt A. Calculada amb el mètode distància d'aquesta classe.

  - u major a 1

    La distància és igual a la distància entre el punt C i el punt B. Calculada amb el mètode distància d'aquesta classe.

  - entre 0 i 1

    La distància és igual a la distància entre el punt C i un punt interior al segment definit per les equacions següents.

longitud = x1 + u (x2 - x1) latitud = y1 + u (y2 - y1)

On x1=longitud del punt A, x2=longitud del punt B, y1=latitud del punt A i y2= latitud del punt B.

- pdis(a, b, c)

  Aquest mètode és igual a l'anterior però, en comptes de calcular la distància entre un segment i un punt, calcula la distància entre un punt i una recta. El càlcul també és el mateix però en aquest cas no és depenent de u i es calcula sempre com en el cas entre 0 i 1.

- minLeftangle(Table)

  Retorna el punt de dins de Table ([[angle0,ID0], [angle1,ID1]...]) que té el menor angle positiu. Els angles negatius es giren sumant-lis 2*pi.

- PointAngularDistanceStartLine(PLStart,PLEnd,ADistance)

  Retorna el punt que és troba a la distància angular donada (ADistance) respecte al punt inicial del vector *PLStart-PLEnd*, veure figura 54.

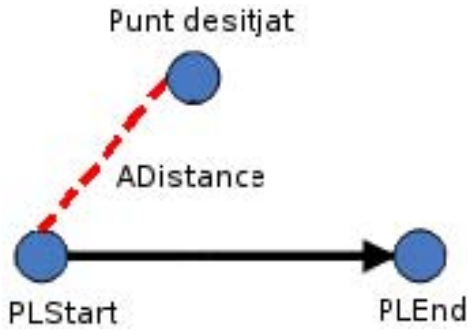


Figura 54: Distància angular

S'utilitza el mètode invers a l'utilitzat al mètode Adistance d'aquesta classe. A continuació es mostren les formules utilitzades.

$$Adist = ADistance$$
$$Lat = LatitudPuntPLEnd - LatitudPuntPLStart$$
$$Lon = LongitudPuntPLEnd - LongitudPuntPLStart$$
$$u = \sqrt{\frac{\overline{V}^2}{Lat^2 + Lon^2}}\ .$$

$$LongitudPuntDesitjat = LongitudPuntPLStart + u * Adist$$
$$LatitudPuntDesitjat = LatitudPuntPLStart + u * Adist$$

- perpendicularVector(Vector)

  Retorna el vector perpendicular al *Vector*. Per calcular-lo simplement es canvia de signe de la latitud i es giren les coordenades.

4.3.10 Classe BiGauss

Aquesta classe s'utilitza per calcular la probabilitat utilitzant una funció Bi-Gaussiana (per més informació anar a l'apartat 3.2.1.1).

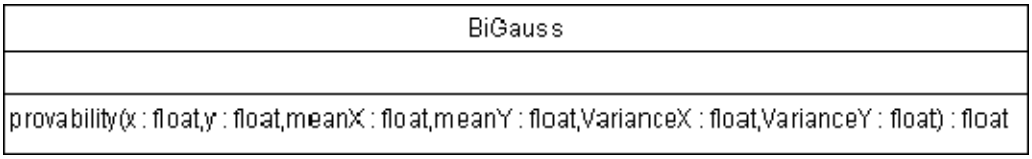


Figura 55: Classe BiGauss

- probability(x,y,meanX,meanY,VarianceX,VarianceY)

  Retorna la probabilitat de l'element amb valors *x* i *y* per a les variables de les que tenim la mitja (*meanX* i *meanY*) i la variança (*VarianceX* i *VarianceY*) utilitzant una funció Bi-Gaussiana (per més informació veure l'apartat 3.2.1.1).

4.3.11 Classe streetGenerator

Aquesta classe és l'encarregada de generar la xarxa de carrers a partir de les estadístiques calculades anteriorment.

Cadascun dels objectes d'aquesta classe guarda 5 diccionaris:

1. L'Statistics on es guarda la id de cada intersecció com a índexs, i un llistat de les estadístiques del punt en aquest ordre :
   - nivell.
   - tortuositat.
   - mitja de la distància.
   - variança de la distància.
   - mitja de l'angle.
   - variança de l'angle.
2. El LevelIntersection on l'índex indica el nivell i les dades són un llistat de totes les interseccions d'aquest nivell. A més, també guarda tres diccionaris més:
3. El diccionari nUsedIntersections que guarda el nombre de segments que arriben/surten de cadascuna de les interseccions
4. El Coordinates on es guarden les coordenades de cadascuna de les interseccions
5. L'Edges i que guarda a cadascuna de les interseccions el llistat d'interseccions a les que es troba connectada.

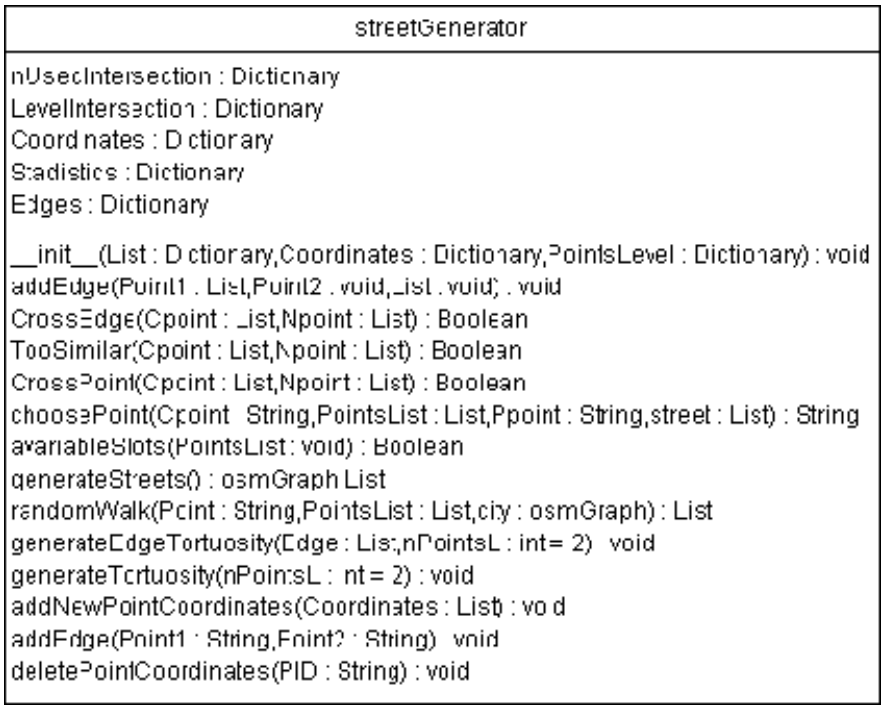


Figura 56: Classe streetGenerator

- __init__(List,Coordinates,PointsLevel)

  Aquest és el mètode constructor. Assigna els paràmetres donats als atributs de l'objecte de la següent manera:

  - A l'atribut Statistics, el paràmetre *List*.
  - A l'atribut LevelIntersection, el paràmetre PointsLevel.
  - A l'atribut Coordinates, el paràmetre Coordinates.

  També inicia l'atribut nUsedIntersection amb els identificadors de les interseccions que es troben a *List* com a índex i un zero com a dada.

  Els paràmetres que es passen a aquest mètode han de tenir el mateix format que els atributs als que s'assignaran. El format d'aquests es troben explicats a la definició d'aquesta classe.

- addNewPointCoordinates(Coordinates)

  Genera una nova id de punt mirant quina és la id actual més alta i sumant-li 1, i l'afegeix al diccionari Coordinates assignant-li el paràmetre *Coordinates* com a dada. Retorna la id creada.

- addEdge(Point1,Point2)

  Si *Point1* no existeix, l'afegeix a l'atribut Edges, tant si ja existia com si s'acaba d'afegir se li afegeix *Point2* com a Intersecció connectada.

- deletePointCoordinates(PID)

  Elimina el punt amb id=*PID* del diccionari Coordinates.

- CrossEdge(Cpoint,Npoint)

  Retrona un booleà que ens indica si l'aresta entre *Cpoint* i *Npoint* creua alguna de les arestes existents.

  Per calcular aquest booleà es fa un recorregut per totes les arestes per comprovar si alguna d'elles creua amb la donada. Per fer això s'utilitza el mètode SegmentIntersection de la classe geo (més informació a l'apartat 4.3.9).

  A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció crossEdge(Cpoint,Npoint):
 2      Cross:=Fals
 3      Edges:= Totes les ids de Edges
 4      per cada aresta edge de Edges fer:
 5          ENpoint:= edge és diferent a Npoint
 6          ECpoint:= edge és diferent a Cpoint
 7          si ENpoint i ECpoint fer:
 8              a:= Coordinates[edge]
 9              Fpoints:=Edges[edge]
10              per cada punt Fpoint de Fpoints fer:
11                  NFpoint:= Fpoint és diferent a Npoint
12                  CFpoint:= Fpoint és diferent a Cpoint
13                  si NFpoint i CFpoint llavors:
14                      b=Coordinates[Fpoint]
15                      c=Coordinates[Cpoint]
16                      d=Coordinates[Npoint]
17                      inters:=geo.SegmentIntersection(a,b,c,d)
18                      si inters llavors:
19                          cross:=True
20                          Sortir
21      retorna cross
```

- TooSimilar(Cpoint,Npoint)

  Retrona un booleà que indica si l'aresta entre *Cpoint* i *Npoint* té un angle més petit que l'indicat per l'atribut MinStreetsAngle de la classe Configuration, respecte a alguna de les arestes existents.

  Per calcular aquest booleà es calcula l'angle que forma l'aresta donada amb les altres arestes que passen per Cpoint i Npoint. Veure figura 57. Per calcular l'angle s'utilitza el mètode angle de la classe geo (més informació a l'apartat 4.3.9).

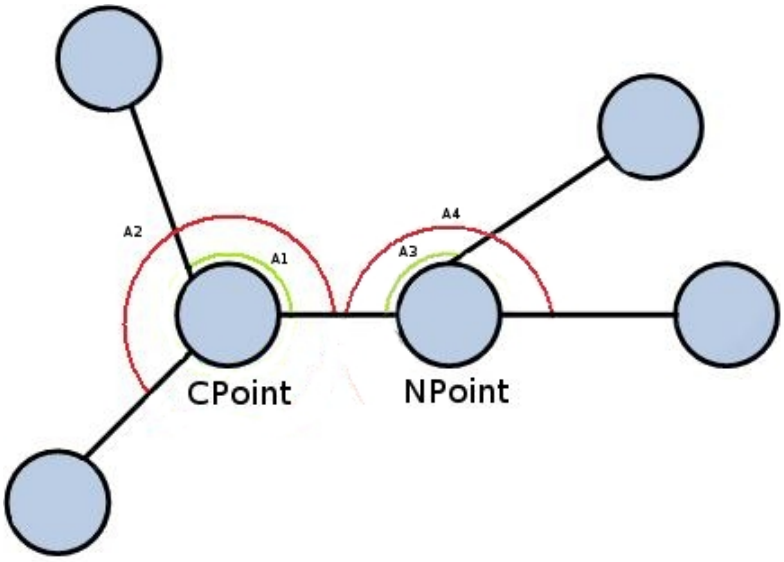


Figura 57: Diagrama angles aresta Cpoint-Npoint

A continuació es mostra el pseudocodi d'aquest algorisme.

```
1   funció tooSimilar(Cpoint,Npoint):
2      Cp:=Np:=Fals
3      minAngle:=Configuration.MinStreetsAngle
4      si Cpoint existeix a Edges llavors:
5          Tedges:=Edges[Cpoint]
6          per cada aresta edge de Tedges fer:
7             CpointC:= Coordinates[Cpoint]
8             NpointC:= Coordinates[Npoint]
9             EpointC:= Coordinates[edge]
10            angle=geo.angle(CpointC,NpointC,EpointC)
11            Amajor:=minAngle és major que angle
12            ANmajor:=angle angle és major que minAngle*-1
13            si Amajor i ANmajor llavors:
14                 Cp:=Cert
15                 Sortir
16     si Npoint existeix a Edges llavors:
17         Tedges:=Edges[Cpoint]
18         per cada aresta edge de Tedges fer:
19            CpointC:= Coordinates[Cpoint]
20            NpointC:= Coordinates[Npoint]
21            EpointC:= Coordinates[edge]
22            angle=geo.angle(NpointC,CpointC,EpointC)
23            Amajor:=minAngle és major que angle
24            ANmajor:=angle angle és major que minAngle*-1
25            si Amajor i ANmajor llavors:
26                 Cp:=Cert
27                 Sortir
28     Ret:= Cp o Np
29     retorna Ret
```

- CrossPoint(Cpoint,Npoint)

  Retrona un booleà que ens indica si l'aresta entre *Cpoint* i *Npoint* passa a una distància menor que l'indicada per l'atribut MinStreetsDistance de la classe Configuration d'algun punt.

Per calcular aquest booleà es calcula la distància entre aquesta aresta i totes les interseccions. Veure figura 58. Per fer això s'utilitza el mètode DPointSegment de la classe geo (més informació a l'apartat 4.3.9).

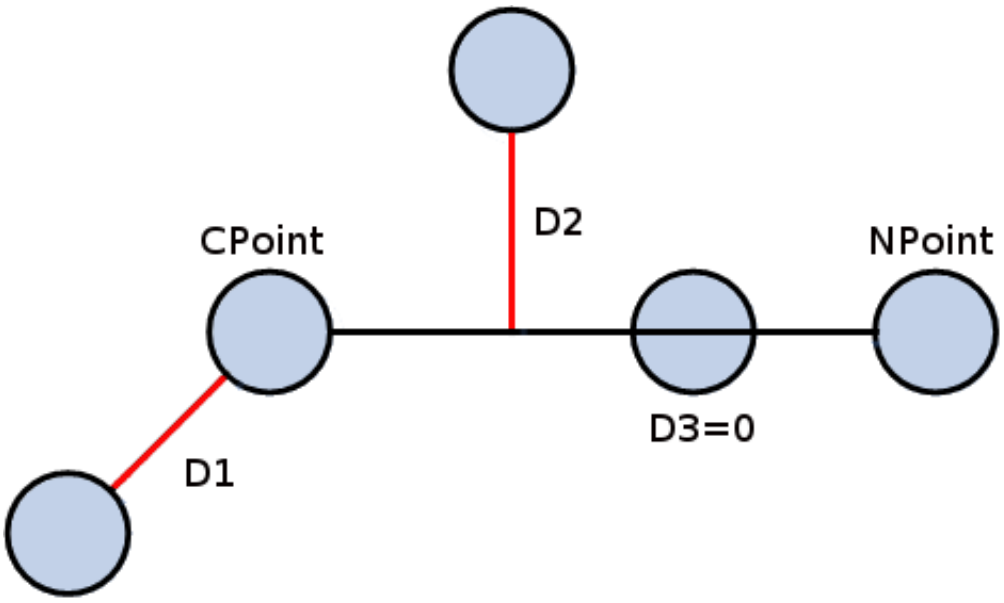


Figura 58: Diagrama distàncies aresta Cpoint-Npoint respecte a les altres interseccions

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció crossPoint(Cpoint,Npoint):
 2      Cross:=Fals
 3      MinDist:=Configuration.MinStreetsDistance
 4      Points:=Totes les ids de Intersection
 5      A:=Coordinates[Cpoint]
 6      b:=Coordinates[Npoint]
 7      per cada punt point de Points fer:
 8          si point és diferent a Cpoint i a Npoint llavors:
 9              c:=Coordinates[point]
10              distance:=geo.DPointSegment(a,b,c)
11              si distance és menor a MinDist llavors:
12                  Cross:=Cert
13                  Sortir
14      retorna  Cross
```

- choosePoint(Cpoint,PointsList,Ppoint,street)

  Per cada intersecció de la llista PointsList, es comprova si l'aresta que l'uniria amb Cpoint :

  - Creua amb una aresta existent (mètode CrossEdge d'aquesta classe)
  - És massa semblant a una aresta existent (mètode TooSimilar d'aquesta classe)
  - Passa massa a prop d'un altre intersecció (mètode CrossPoint d'aquesta classe)

- La intersecció encara admet que se li afegeixin arestes (el valor corresponent a ella a nUsedIntersection és menor que MaxSegmentsIntersection)

- La intersecció ja forma part del carrer (si existeix dins la llista *street*)

Si es compleixen totes aquestes condicions, es procedeix a calcular la probabilitat d'aquesta intersecció. Si alguna de les condicions anteriors no es compleix, el punt es descarta.

Per calcular la probabilitat primer es calcula:

- la distància entre Cpoint i la intersecció (punt_actual)

- l'angle entre els vectors Ppoint-Cpoint

- l'angle entre els vectors Cpoint-punt_actual

Un cop es tenen aquestes dades es crida el mètode probabilitat de la classe BiGauss (més informació a l'apartat 4.3.10). Si la probabilitat resultant és superior a MinSegmentProvability (atribut de la classe Configuration), el punt s'afegeix a una llista de punts possibles. Si no el punt és descartat.

Un vegada s'ha obtingut la llista de punts possibles aquesta es passa com a paràmetre al mètode cdf de la classe CDF (més informació a l'apartat 4.3.8). El punt que retorna el mètode cdf és el mateix que retornarà aquest mètode.

A continuació es mostra el pseudocodi d'aquest algorisme.

```
 1   funció choosePoint(Cpoint,PointsList,Ppoint,street):
 2      SelectablePoints:=sPoints:=Llista Buida
 3      Npoint:=Nul
 4      MaxA:=Configuration.MaxSegmentAngle
 5      MaxL:=Configuration.MaxSegmentLenght
 6      MaxSI:=MaxSegmentsIntersection
 7      MinProb:=Configuration.MinSegmentProvability
 8      per cada punt Point de PointsList fer:
 9          creua:=crossEdge(Cpoint,Point)
10          Clon:=Coordinates[Cpoint][0]
11          Clat:=Coordinates[Cpoint][1]
12          Plon:=Coordinates[Point][0]
13          Plat:= Coordinates[Point][1]
14          distance=geo.distance(Clon,Clat,Plon,Plat)
15          si Ppoint és Nul fer:
16              angle:=Statistics[Cpoint][3]
17          altrament:
18              PpointCoords:=Coordinates[Ppoint]
19              CCoords:=Coordinates[Cpoint]
20              PCoords:=Coordinates[Point]
21              angle=geo.angle(CCoords, PpointCoords ,PCoords)
22          si angle és menor que 0 llavors:
23              angle:=2*pi+angle
24          inv:=(math.pi*2)- MaxA
25          goodangle:= angle més gran que MaxA
26          goodangle:= goodangle i angle més petit que inv
27          si MaxL és 0 o distance és menor que MaxL llavors:
28              usat:= nUsedIntersection[Point]
29              CFree:= usat és menor que MaxSI
```

```
30              Dif:= Point és diferent a Cpoint
31              tS:= tooSimilar(Cpoint,Point)
32              cross:= crossPoint(Cpoint,Point)
33              aux:= Dif i CFree i no creua
34              si aux i no tS i no cross i goodangle:
35                  d:= distance
36                  a:=angle
37                  DM:= Statistics[Cpoint][“DistanceMean”]
38                  AM:= Statistics[Cpoint][“AngleMean”]
39                  DV:= Statistics[Cpoint][“DistanceVariance”]
40                  AV:= Statistics[Cpoint][“AngleVariance”]
41                  Prob=BiGauss.probability(d,a,DM,AM,DV,AV)
42                  GProb:= Prob és major que MinProb
43                  Pe:=Point existeix a sPoints
44                  PeS:=Point existeix a street
45                  si GProb i no Pe i no PeS llavors:
46                      SelectablePoints.Afegeix([Prob,Point])
47                      sPoints.Afegeix(Point)
48      Npoint:=CDF.cdf(SelectablePoints)
49      retorna Npoint
```

- avariableSlots(PointsList)

  Retorna un booleà que indica si per algun dels punts (*PointsList*) el valor emmegatzemat a nUsedIntersections per aquest punt és menor a l'atribut MaxSegmentsIntersection de la classe Configuration

- GenerateStreets()

  Aquest mètode és l'encarregat de generar la nova xarxa de carrers i retorna un objecte de la classe osmGraph i un llistat amb les interseccions que formen cada carrer.

  Per aconseguir aquest resultat, el mètode recorre totes les interseccions d'un nivell i crida per a cadascuna d'elles el mètode randomWalk (mirar mètode següent) amb la intersecció actual com al paràmetre Point, el llistat de les Interseccions d'aquest nivell com a Points List i un objecte osmGraph com a city. El mètode canvia de nivell quan ja no es poden afegir més arestes als punts del nivell o quan s'ha fet un recorregut complert per totes elles i no s'ha pogut afegir cap aresta. Un cop s'han recorregut tots els nivells el mètode retorna l'objecte osmGraph i el llistat dels carrers.

  A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció GenerateStreets():
 2       City:=nou osmGraph('')
 3       city.addPoints(llista ids de Coordinates,Coordinates)
 4       levels:= llista ids de LevelIntersection
 5       levels:=ordena levels de major a menor
 6       Nstreets:=llista buida
 7       per cada nivell level de levels fer:
 8           Points:=llista ids de level a LevelIntersection
 9           Continue:=cert
10           mentre avariableSlots(Points) i Continue fer:
11              Continue:=fals
12              Per cada punt point de Points fer:
13                  AuxS=self.randomWalk(point, Points ,city)
```

```
14                  si hi ha més d'un punt a Auxstreets llavors:
15                      Continue:=Cert
16                      Nstreets.afegir(AuxS)
17                      city.addStreet(AuxS)
18      retorna city i Nstreets
```

- randomWalk(Point,PointsList,city)

  Aquest mètode genera un carrer amb inici a *Point* i possibles punts *PointsList*. Llavors afegeix punts al carrer per generar la tortuositat, afegeix el carrer a l'objecte osmGraph *city* i el retorna.
  Per aconseguir aquests resultats, el mètode crida el mètode choosePoint d'aquesta mateixa classe amb el punt actual com a Cpoint, juntament amb PointsList i el llistat de punts del carrer que estem generant com a street. Una vegada ja s'ha escollit el punt següent, es creen els punts per generar la tortuositat. Aquests punts estan determinats per l'atribut TortuosityPoints de la classe configuration, cridant el mètode generateEdgeTortuosity d'aquesta classe. El mètode generateEdgeTortuosity pot crear punts de tortuositat en sentit positiu o negatiu, el mètode randomWalk el crida de tal manera que un segment els tingui positius i el següent Negatius, veure Figura 59.

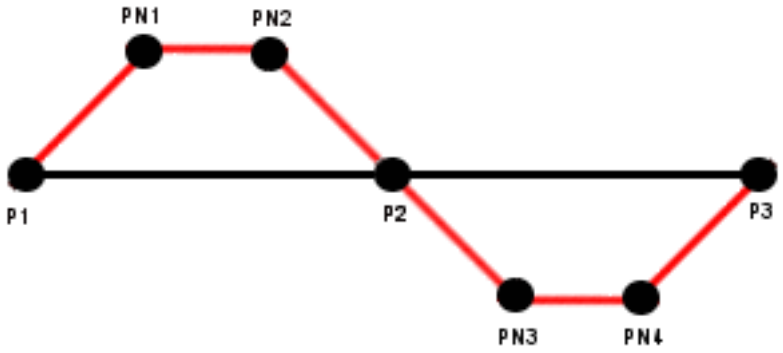


Figura 59: Diagrama segments consecotius Carrer, un amb tortuositat positiva i l'altre negativa

Quan ja es tenen tots els punts, s'afegeixen a *city* i al llistat de punts del carrer. El mètode no finalitza un carrer fins que el mètode choosePoint no retorna cap intersecció o quan totes les Interseccions de *PointsList* ja no admeten més segments.

A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció randomWalk(Point,PointsList,city):
 2      PL:= PointsList
 3      TP:= Configuration.TortuosityPoints
 4      gET:=generateEdgeTortuosity     //defineix un alies
 5      Stop:= Fpoint:=False
 6      Positive:=Cert
 7      street:=llista buida
 8      Flenght:=0
 9      si hi ha més d'un element a PointsList llavors:
10          Cpoint=Point
11          Ppoint=Nul
12          Street:= una llista amb només Cpoint
13          used:=valor guardat a nUsedIntersection per a Cpoint
14          mentre no stop i used menor que 4 fer:
15              Npoint=choosePoint(Cpoint,PL,Ppoint,street)
16              si Npoint és Nul llavors:
17                  stop:=cert
```

```
18                altrament:
19                    Plist:= gET([Cpoint, Npoint],positive,TP)
19                    positive:= no positive
19                    Start:=Cpoint
20                    Per cada punt Point de Plist fer:
21                        p:=city.addNewPoint((Point[0],Point[1]))
22                        city.addEdge(start, p)
23                        street.afegir(p)
24                        start:=p
25                    street.afegir(Npoint)
26                    city.addEdge(start, Npoint)
27                    +1 al valor de Cpoint a nUsedIntersection
28                    +1 al valor de Npoint a nUsedIntersection
29                    Ppoint:=Cpoint
30                    Cpoint:=Npoint
31        retorna street
```

- generateEdgeTortuosity(Edge,positive, nPoints=2, SecondTry=False)

  Aquest mètode genera un o dos punts (*nPoints*) per tal de dotar l'aresta (*Edge*) amb la tortuositat mitjana de les dos interseccions que la formen. Depenent del valor de la variable positive els punts es creen en sentit positiu, veure figura 59. En cas contrari es creen en sentit invers, veure figura 59. Per calcular aquest punts primer es calcula la distància a la que es trobarà el punt a generar respecte a l'aresta que formen les dos Interseccions i després es crida el mètode PointAngularDistanceStartLine de la classe geo.

  Per calcular la distància entre el punt/s i l'aresta (H) s'utilitzen els següents supòsits i formules:

  $DTT = L * T$ on DTT és la distància de tortuositat, L és la distància entre els dos punts i T és la tortuositat mitjana d'aquests dos punts.

  - Cas 1 punt(veure codi abaix, entre les línies 73 i 112)

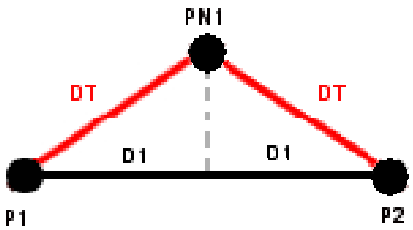


Figura 60: Diagrama distàncies càlcul un punt tortuositat

$$D1 = D2 = \frac{L}{2}$$

$$DT = \frac{DTT}{2}$$

$$H = \sqrt{\left| D1^2 + DT^2 - 2 * D1 * DT * \frac{D1}{DT} \right|}$$

o Cas 2 punts (veure codi abaix, entre les línies 18 i 72)

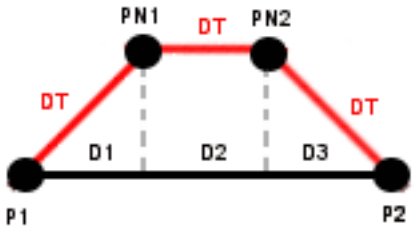


Figura 61: Diagrama distàncies càlcul dos punts tortuositat

$$DT = \frac{DTT}{3}$$

$$D2 = DT$$

$$D1 = D3 = \frac{L - D2}{2}$$

$$H = \sqrt{\left| D1^2 + DT^2 - 2 * D1 * DT * \frac{D1}{DT} \right|}$$

Un cop s'han creat els punts demanats, es comprova que les arestes que els uneixen amb els punts originals no creuin amb cap altre aresta
A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció generateEdgeTortuosity(Edge,positive, nPoints=2,
                                  SecondTry=False):
 2      PADSL:=geo.PointAngularDistanceStartLine
 3      gET:=generateEdgeTortuosity
 4      p:=positive
 5
 6      Elon:= Coordinates[Edge[0]][0]
 7      Elat:= Coordinates[Edge[0]][1]
 8      E2lon:= Coordinates[Edge[1]][0]
 9      E2lat:= Coordinates[Edge[1]][1]
10      Lm=geo.distance(Elon,Elat, E2lon,E2lat)
11      T:=Statistics[Edge[0]]["Tortuosity"]
12      T:=(T+Statistics[Edge[1]]["Tortuosity"])/2
13      Dm=Lm*T
14      L=geo.aDistance(Elon,Elat,E2lon,E2Lat)
15      D=L*T
16      Plist:=Llista Buida
17
18      si nPoints és 2 fer:
19          bsd:=D/3
20          ssd:=(L-(bsd))/2
21          si bsd no és 0 fer:
22              H:=√|ssd  + bsd^2 - 2*ssd*bsd*ssd/bsd|
23              EC:= Coordinates[Edge[0]]
24              E2C:= Coordinates[Edge[0]]
25
26              point1:=geo.PADSL(EC,E2C,ssd)
```

```
27              point2:=geo.PADSL(E2C,EC,ssd)
28              si positive fer:
29                 Vect1=(point1[0]-Elon,point1[1]-Elat)
30                  Vect2=(E2lon-point2[0],E2lat-point2[1])
31              altrament:
32                Vect1=(Elon-point1[0],Elat-point1[1])
33                  Vect2=(point2[0]-E2lon,point2[1]-E2Lat)
34
35              PVect1:=geo.perpendicularVector(Vect1)
36              PVect2:=geo.perpendicularVector(Vect2)
37              Paux1Lon:= point1[0]+PVect1[0]
38              Paux1Lat:= point1[1]+PVect1[1]
39              Paux1=(Paux1Lon,Paux1Lat)
40              Paux2Lon:= point2[0]+PVect2[0]
41              Paux2Lat:= point2[1]+PVect2[1]
42              Paux2=(Paux2Lon,Paux2Lat)
43              P1=geo.PADSL(point1,Paux1,H)
44              P2=geo.PADSL(point2,Paux2,H)
45              p1=addNewPointCoordinates(P1)
46              p2=addNewPointCoordinates(P2)
47
48              p1E0:=crossEdge(Edge[0],p1)
49              p1E0:=p1E0 o crossPoint(Edge[0],p1)
50              p1p2:=crossEdge(p1,p2) o crossPoint(p1,p2)
51              p2E1:=crossEdge(p2,Edge[1])
52              p2E1:=p2E1 o crossPoint(p2,Edge[1])
54
55              si p1E0 o p1p2 o p2E1 llavors:
56                  deletePointCoordinates(p1)
57                  deletePointCoordinates(p2)
58                  si SecondTry llavors:
59                      addEdge(Edge[0],Edge[1])
60                      addEdge(Edge[1],Edge[0])
61                      Plist:=Llista Buida
62                  altrament:
63                      Plist:=gET(Edge,p,nPoints,Cert)
64              altrament:
65                  addEdge(Edge[0],p1)
66                  addEdge(p1,Edge[0])
67                  addEdge(p1,p2)
68                  addEdge(p2,p1)
69                  addEdge(p2,Edge[1])
70                  addEdge(Edge[1],p2)
71                  Plist:=[P1,P2]
72
73      altresi nPoints és 1 fer:
74          ssd:=L/2
75          bsd:=D/2
76          si bsd no és 0 fer:
77                  H:=\sqrt{\left| ssd \ + bsd^2 - 2*ssd*bsd*\frac{ssd}{bsd} \right|}
78              EC:= Coordinates[Edge[0]]
79              E2C:= Coordinates[Edge[0]]
80            point1:=geo.PADSL(EC,E2C,ssd)
81
82              si positive fer:
83                  Vect1=(point1[0]-Elon,point1[1]-Elat)
84              altrament:
85                  Vect1=(Elon-point1[0],Elat-point1[1])
86
```

```
87              PVect1:=geo.perpendicularVector(Vect1)
88             Paux1Lon:= point1[0]+PVect1[0]
89              Paux1Lat:= point1[1]+PVect1[1]
90              Paux1=(Paux1Lon,Paux1Lat)
91              P1=geo.PADSL(point1,Paux1,H)
92              p1=addNewPointCoordinates(P1)
93
94              p1E0:=crossEdge(Edge[0],p1)
95              p1E0:=p1E0 o crossPoint(Edge[0],p1)
96              p1E1:=crossEdge(p1,Edge[1])
97              p1E1:=p1E1 o crossPoint(p1,Edge[1])
98
99              si p1E0 o p1E1 llavors:
00                  deletePointCoordinates(p1)
01                  si SecondTry llavors:
02                      addEdge(Edge[0],Edge[1])
03                      addEdge(Edge[1],Edge[0])
04                      Plist:=Llista Buida
05                  altrament:
06                      Plist:=gET(Edge,p,nPoints,Cert)
07              altrament:
08                  addEdge(Edge[0],p1)
09                  addEdge(p1,Edge[0])
10                  addEdge(p1,Edge[1])
11                  addEdge(Edge[1],p1)
12                  Plist:=[P1]
13      altament:
14          addEdge(Edge[0],Edge[1])
15          addEdge(Edge[1],Edge[0])
16          Plist:=Llista Buida
17
18      retorna Plist
```

- generateTortuosity(nPoints=2)

  Genera la tortuositat per totes les arestes. Per fer això simplement crida el mètode generateEdgeTortuosity per a cadascuna de les arestes emmagatzemades a l'atribut Edges. Els punts de tortuositat són generats de tal manera que una aresta els té positius i la següent Negatius, veure figura 59.

  A continuació es mostra el pseudocodi d'aquest mètode.

```
1  funció generateTortuosity(nPoints=2):
2      gET:=generateEdgeTortuosity
3      Edges=ids de Edges
4      per cada aresta edge de Edges fer:
5          Points:=Edges[edge]
6          Positive:=Cert
7          per cada punt point de Points fer:
8              gET([edge,point],positive,nPoints)
9              positive:= no positive
```

4.3.12 Classe houdiniOSM

Aquesta classe agrupa tots els mètodes que transformen les dades provinents de les classes anteriors en dades per poder treballar amb el Houdini.

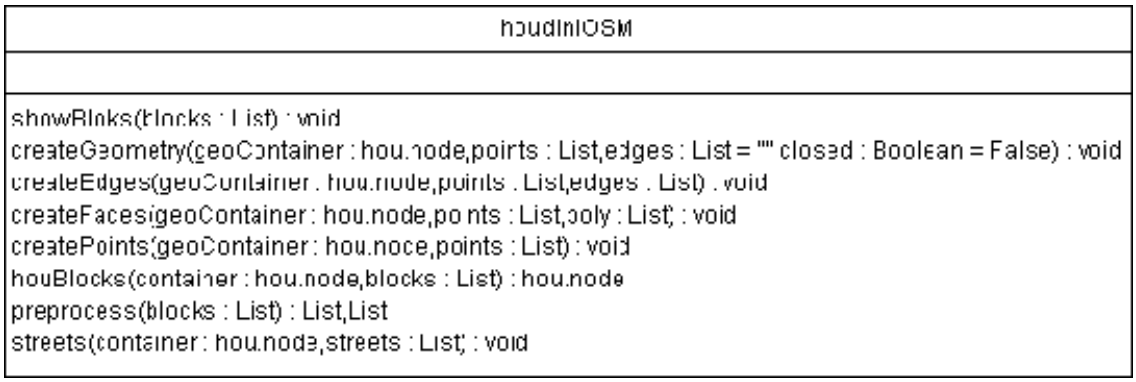


Figura 62: Classe houdiniOSM

- showBloks(blocks)

  Retorna una cadena amb les coordenades de cada illa (*blocks*)

- createGeometry(geoContainer, points, edges="", closed=False)

  Genera els nodes Houdini necessaris per dibuixar l'objecte representat pels punts (points), les arestes (edges) i blocks (si les arestes formen un polígon tancat) (closed) i els guarda dins de *geoContainer.*

  Primer s'afegeixen tots els punts del llistat (*points*) a l'estructura de nodes (*geoContainer*) convertint-los en sistema de coordenades de tres dimensions on la tercera sempre és 0. Un cop s'ha finalitzat aquest procés és creen les unions entre els punts (*arestes*). Retorna un node del tipus Null.

  A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció createGeometry(geoContainer,points,edges="",c=Fals):
 2     geo:=geoContainer.createNode("add")
 3     null:=geoContainer.createNode("null", "HOM_Geo")
 4     lp:=nº d'elements a points
 5     le:=nº d'elements a edges
 6     geo.parm("points").set(lp)
 7     eo.parm("prims").set(le)
 8     ptlist:=Llista de 0 a lp
 9     per cada nombre ptnum de ptlist:
10        usept:= "usept" + str(ptnum)
11        pt:= "pt"+str(ptnum)
12        geo.parm(usept).set(1)
13        geo.parm(pt+"x").set(points[ptnum][0])
14        geo.parm(pt+"y").set(points[ptnum][1])
15        geo.parm(pt+"z").set(points[ptnum][2])
16
17     edgelist:=Llista de 0 a le
18     per cada edge de edgelist:
19        pr := "prim" + str(edge)
20        strE:= Convertir el elements de edges[edge]] a strings
21        geo.parm(pr).set(strE)
```

```
22
23      si closed fer:
24        pr_closed := "closed"+str(edge)
25        geo.parm(pr_closed).set(1)
26
26      null.setFirstInput(geo)
28      ull.setHardLocked(1)
29      geo.destroy()
30      retorna null
```

- createEdges(geoContainer, points, edges)

  Genera els nodes Houdini necessaris per dibuixar les línies representades per *points* i *edges*. Un cop generats, es guarden a *geoContainer*. Per generar aquest nodes s'utilitza el mètode anterior. Retorna un node del tipus Null.

- createFaces(geoContainer, points, poly)

  Genera els nodes Houdini els necessaris per dibuixar els polígons tancats que es troben descrits per *points* i *poly*. Un cop generats es guarden a *geoContainer*. Per generar aquest nodes s'utilitza el mètode createGeometry amb *poly*=edges i closed=True. Retorna un node del tipus Null.

- createPoints(geoContainer, points)

  Genera els nodes Houdini que representen (*points*) i els guarda a *geoContainer*. Per generar aquest nodes s'utilitza el mètode createGeometry deixant closed i edges amb el seu valor per defecte. Retorna un node del tipus Null.

- preprocess(blocks)

  Retorna dos llistes una amb les coordenades expressades en sistema de 3 coordenades i l'altre amb els identificadors de tots els punts que formen les illes.

  A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció preprocess(blocks):
 2     polys:=verts:=Llista Buida
 3     currVert:=0
 4     per cada camí path de blocks fer:
 5       face:=Llista Buida
 6       per cada parellCordenades (coordx,coordy) de path fer:
 7               x = coordx
 8               y = coordy
 9               verts.Afegir( (x,y,0) )
10               face.Afegir(currVert)
11               currVert:= currVert+1
12       polys.Afegir(face)
13     retorna verts, polys
```

- blocks(container, blocks)

  Rep un node Houdini (*container*) i un llistat de coordenades que representen les illes(*blocks*) i retorna un node Houdini amb la geometria necessària perquè

es puguin dibuixar les illes. Qualsevol dades que estigui guardada anteriorment a *container* serà eliminada durant el procés.
A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció blocks(container, blocks):
 2     per cada fill n de container.children() fer:
 3       n.destroy()
 4     verts, polys:= preprocess(blocks)
 5     geometry:=createFaces(container, verts, polys)
 6     dele :=container.createNode('delete')
 7     dele.parm('pattern').set('0')
 8     dele.setFirstInput(geometry)
 9     poli :=container.createNode('polyextrude')
10     poli.setName('blockInitialResize')
11     poli.setFirstInput(dele)
12     poli.parm('inset').set(-0.0001)
13     poli.parm('outputfront').set(1)
14     poli.parm('outputback').set(0)
15     poli.parm('outputside').set(0)
16
17     poli.setDisplayFlag(1)
18
19     per cada fill n de container.children():
20       n.moveToGoodPosition()
21
22     retorna poli
```

- streets(container, streets)

  Rep un node Houdini(*container*) i un llistat de coordenades que representen els carrers(*streets*) i retorna un node Houdini amb la geometria necessària perquè es puguin dibuixar. Qualsevol dada que estigui guardada a *container* serà eliminada durant el procés.

  A continuació es mostra el pseudocodi d'aquest mètode.

```
 1  funció streets(container, streets):
 2     per cada fill n de container.children():
 3        n.destroy()
 4     verts, polys:=preprocess(streets)
 5     geometry:=createEdges(container, verts, polys)
 6
 7     per cada fill n de container.children():
 8        n.moveToGoodPosition()
 9     return geometry
```

4.3.13 Classe urbanEngineNode

Aquesta classe s'utilitza per enganxar aquesta aplicació amb l'urbanEngine.

Cadascun dels objectes d'aquesta classe té quatre atributs:

- el geo on es guarda el contenidor de tots els nodes Houdini
- el first on es guarda el primer node Houdini d'aquest mòdul.
- el last on es guarda l'últim d'aquest mòdul

- el createMerge que indica si el node permet més d'una entrada.

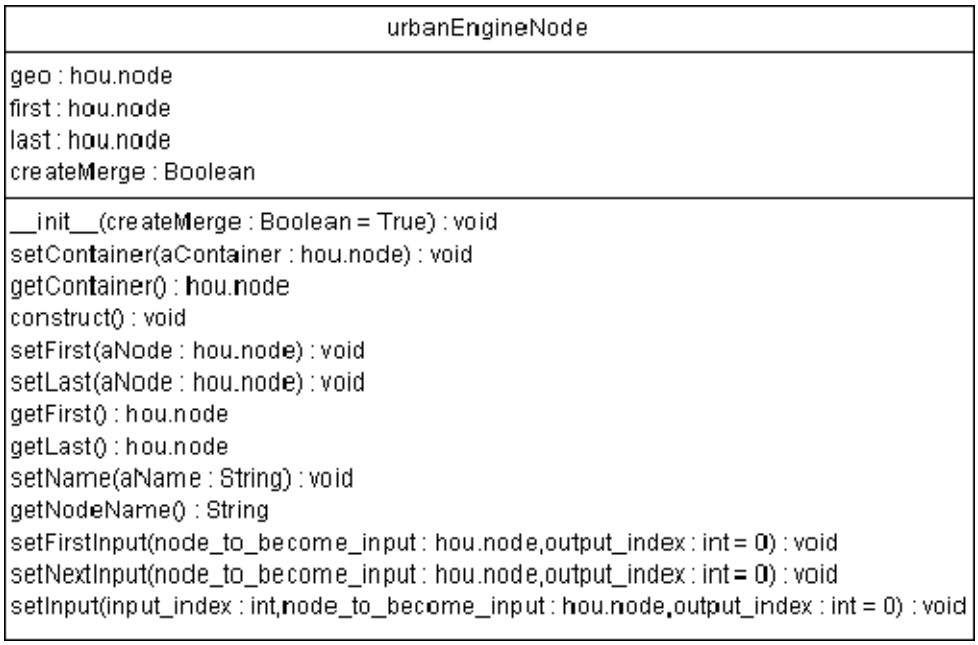


Figura 63: Classe urbanEngineNode

- __init__(createMerge = True)

  Assigna el paràmetre *createMerge* a l'atribut createMerge.

- setContainer(aContainer)

  Assigna el node Houdini *aContainer* com a geo.

- getContainer()

  Retorna geo.

- construct()

  Es tracta d'un mètode virtual per ser implementat en els fills d'aquesta classe. En els fills genera els node Houdini que representen un mòdul urbanEngine.

- setFirst(aNode)

  Assigna a first el node Houdini *aNode*.

- setLast(aNode)

  Assigna a last el node Houdini *aNode*.

- getFirst()

  Retorna el node Houdini first.

- getLast()

  Retorna el node Houdini last.

- setName(aName)

  Assigna el nom *aName* al node last

- getNodeName()

  Retorna l'string urbanEngineNode per tal de diferenciar-lo d'altres nodes urbanEngine.

- setFirstInput(node_to_become_input, output_índex=0)

  Uneix el node final de *node_to_become_input* amb el first de l'objecte actual, afegint el node final *node_to_become_input* com a entrada de first de l'objecte actual .

- setNextInput(node_to_become_input, output_índex=0)

  Afegeix el node final de *node_to_become_input* com a entrada al first de l'objecte actual

- setInput(input_índex, node_to_become_input, output_índex=0)

  Afegeix el node final de *node_to_become_input* com a entrada al node que es troba a la posició *input_índex* de l'objecte actual.

4.3.14 Classe uE_EBStreetGeneratorMajorRoads

Classe que hereta de urbanEngineNode, es tracta d'una especificació de la classe mare per poder treballar millor amb dades provinents de fitxers OSM.
L'atribut blocks guarda una llista de les coordenades dels punts que formen les illes.

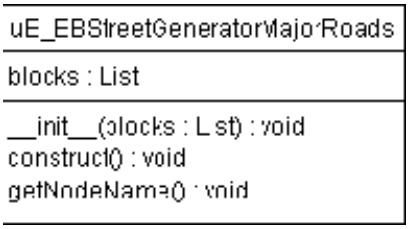


Figura 64: Classe uE_EBStreetGeneratorMajorRoads

- __init__(blocks)

  Aquest mètode és el constructor, rep un llistat amb les coordenades de les illes (*blocks*) i l'assigna a blocks.

- construct()

  Implementació del mètode virtual de la classe mare, es generen tots els nodes per representar les illes. Per fer això es crida el mètode blocks de la classe houdiniOSM.

- getNodeName()

  Retorna l'string uE4DCitiesMajorRoads per tal de diferenciar-lo d'altres nodes Houdini.

4.3.15 Classe uE_EBStreetGeneratorFactory

Aquesta classe és l'encarregada de generar tots els nodes urbanEngine per a construcció de la ciutat. Cadascun dels objectes d'aquesta classe guarda un node contenidor de Houdini a l'atribut geo i una llista llista de les coordenades dels punts que formen les illes.

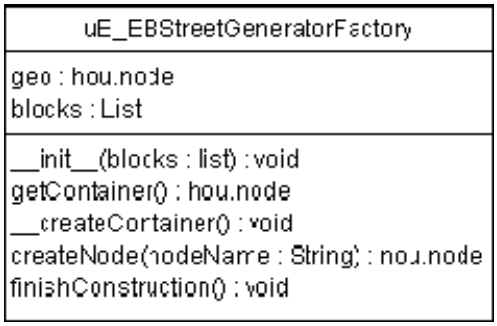


Figura 65: Classe uE_EBStreetGeneratorFactory

- __init__(blocks)

  Aquest mètode és el constructor, crea un node Houdini i l'assigna a geo, també guarda *blocks* a blocks

- getContainer()

  Retorna geo

- __createContainer()

  crea un node Houdini i l'assigna a geo

- createNode(nodeName)

  Depenent dels possibles valors de *nodeName* retorna un Node Houdini amb tota la geometria necessària per dibuixar-lo, el valors possibles són:

  - majorRoads

    Retorna un conjunt de Nodes Houdini que descriu els carrers.

  - minorRoads

    Aquest valor és vàlid però dins d'aquesta aplicació retorna un node de tipus Null. La seva idea és generar carrers secundaris.

  - blocks

    Retorna una llista de Nodes Houdini on s'han eliminat les illes massa petites com perquè hi càpiga un bloc d'edificis.

  - lots

    Genera les parcel·les de les illes i retorna els nodes Houdini que les descriuen.

- buildings

  Genera un edifici per cada parcel·la i retorna els nodes Houdini que les descriuen.

- finalAppearance

  Afegeix les textures als edificis i retorna els nodes Houdini que les descriuen.

- floor

  En aquesta aplicació retorna un node de tipus Null.

- filter

  En aquesta aplicació retorna un node de tipus Null.

- finishConstruction()

  Dona l'ordre al Houdini perquè dibuixi tots els nodes que es troben dins de geo.

4.3.16 Classe network

Aquesta classe marca la seqüència per a la creació de ciutats en 3D de l'urbanEngine, veure figura 67.

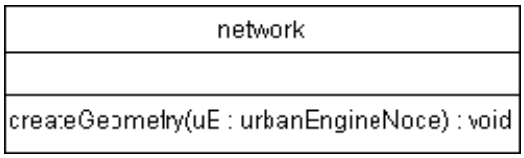


Figura 66 Classe network

- createGeometry(uE)

  Rep un objecte del tipus factory (*uE*), que en aquesta aplicació es tracta d'un objecte de uE_EBStreetGeneratorFactory, i seguint la seqüència de la figura 67 li va demanant a *uE* que generi els nodes corresponents.

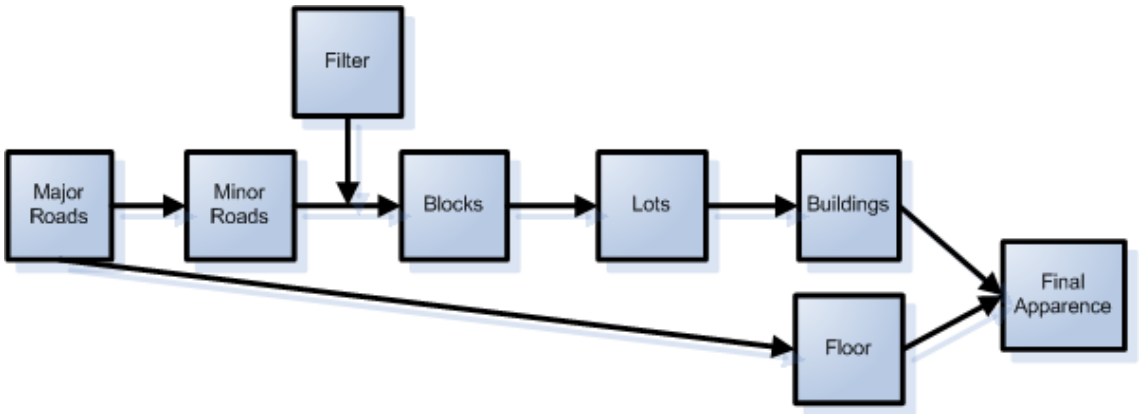


Figura 67: Seqüencia creació de la geometria de la ciutat.

4.4 Diagrames de seqüència

Els diagrames de seqüència presenten el flux de missatges entre diverses instàncies al llarg del temps. Ens permeten veure immediatament l'ordre en que s'envien els missatges perquè el temps es una dimensió explícita del diagrama (eix vertical). Cada instància es representen mitjançant un recorregut vertical diferent.

4.4.1 Diagrama Seqüència generar xarxa de carrers

El diagrama que es mostrarà a continuació mostra com l'usuari li indica a la UI (s'explica a l'apartat 6.1), que actua com a classe de control, que vol generar una xarxa de carrers. Aquesta li demana a la classe dataHandler les estadístiques del mapa OSM carregat. La classe dataHandler demana les dades necessàries per crear un objecte de la classe osmStadistics a l'objecte osmTree que té carregat el mapa OSM. Un cop els hi ha proporcionat crea l'objecte osmStadistics i li demana que calculi les estadístiques del mapa.
Un cop la UI té les estadístiques del mapa, crea un objecte StreetGenerator utilitzant aquestes dades i li demana que generi la xarxa de carrers. Aquest crea un objecte osmGraph i hi va afegint els carrers a mesura que els va creant fins que acaba de crear-los.

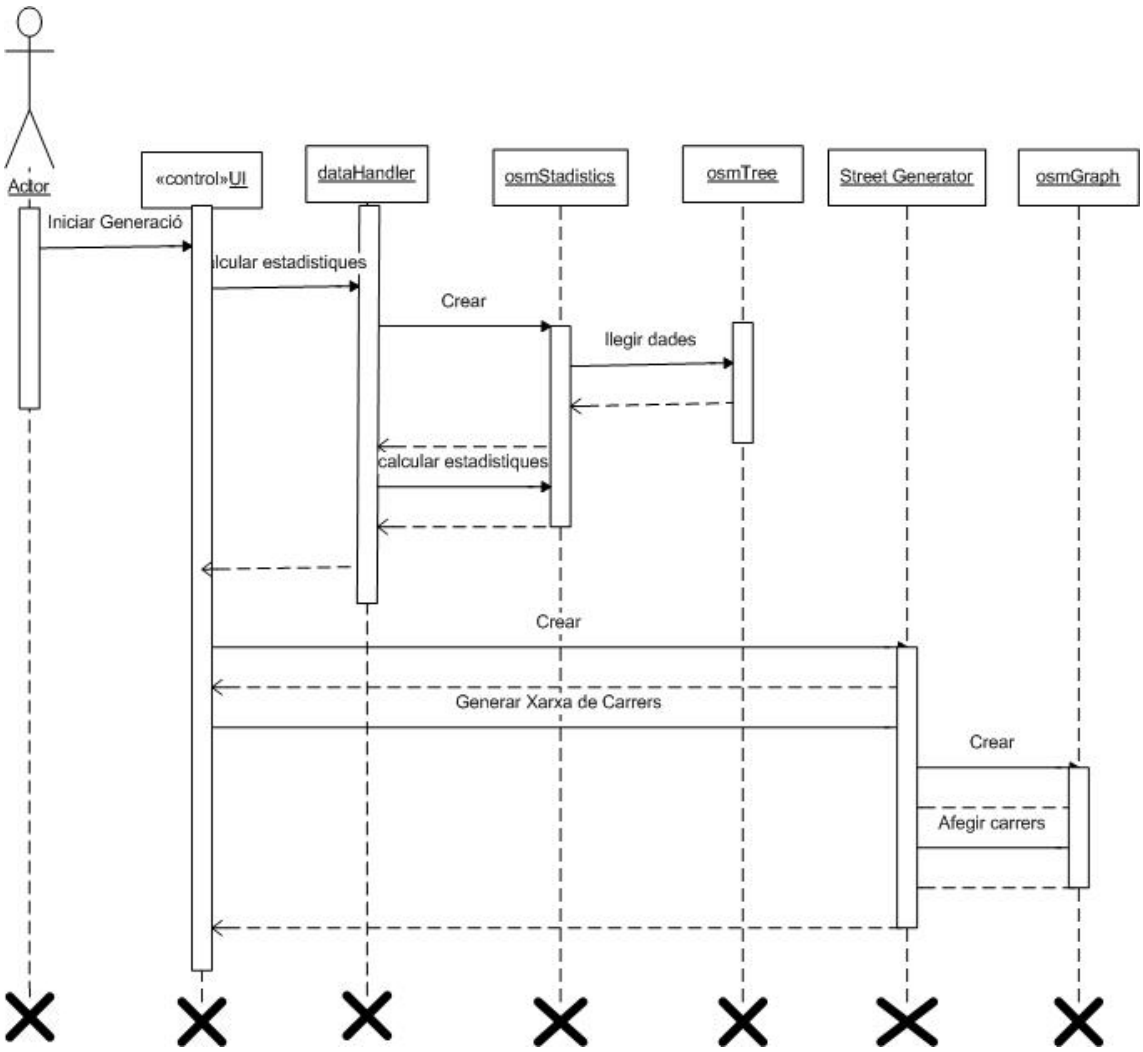


Figura 68 :Diagrama de seqüència generar xarxa de carrers

4.4.2 Diagrama Seqüència generar ciutat 3D

El diagrama que es mostrarà a continuació mostra com l'usuari li indica a la UI (s'explica a l'apartat 6.1), que actua com a classe de control, que vol veure la xarxa de carrers en 3D. Aquesta li demana a la classe dataHandler les illes de la xarxa de carrers generada. Aquesta classe li demana a l'objecte de la classe osmGraph que té guardada aquesta xarxa de carrers que li retorni cadescuna de les illes de la xarxa. Un cop la UI té les illes crea un objecte uE_EBStreetGeneratorFactory i li demana que generi la vista 3D d'aquestes illes. Per fer això aquesta classe crea un objecte uE_EBStreetGeneratorMajorRoads que retorna la geometria per visualitzar la xarxa de carrers en 3D.

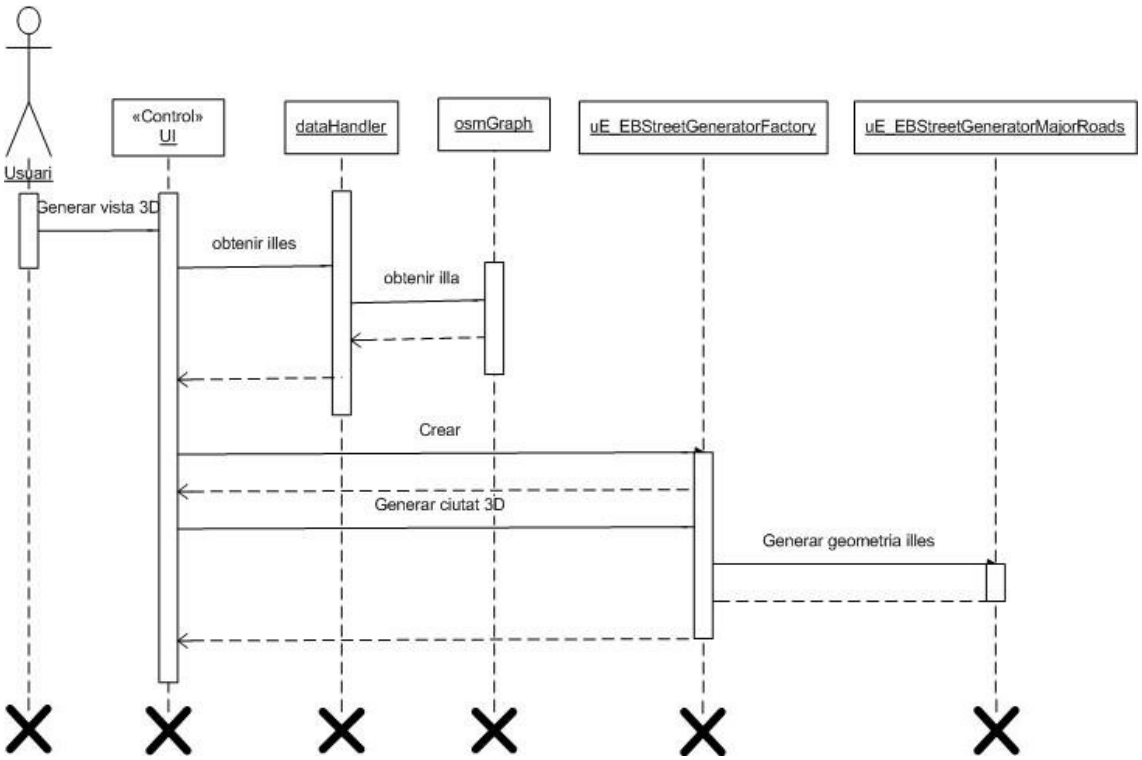


Figura 69 :Diagrama de seqüència generar ciutat 3D

6- Implementació

6.1 UI

La interfície d'usuari (UI) ha estat implementada utilitzant les llibreries wxPython, les classes que s'han utilitzat ha estat:

- wxFrame

  Un objecte d'aquesta classe representa una finestra en que la seva posició i mida poden ser canviades per l'usuari

- wxDialog

  Un objecte d'aquesta classe representa una finestra en que només la seva posició pot ser canviada per l'usuari

- wxTextCtrl

  Un objecte d'aquesta classe representa un quadre de text on aquest pot ser editat, pot contenir més d'una línia.

- wxButton

  Un objecte d'aquesta classe representa un botó que conté un string (normalment el nom de la funció) i que quan és clickat amb el ratolí genera l'event wxEVT_COMMAND_BUTTON_CLICKED que pot ser configurat per executar una funció qualsevol.

- wxBoxSizer

  Els objectes d'aquesta classe s'utilitzen per poder definir la posició dels elements que s'afegeixen a un panell, frame, dialog,... . Aquest objectes actuen dividint l'objecte desitjat (elements del tipus finestra, panell o una divisió d'un altre sizer) horitzontalment o verticalment en el nombre de divisions desitjades.
  A diferència d'altres llibreries, per a la generació de UIs a les wxPython no s'admet assignar la posició directament als elements. Per situar-los on es desitja s'utilitzen els elements de tipus sizer (en aquesta aplicació només s'han utilitzat els tipus wxBoxSizer i wxGridSizer) i elements wxSpacer.

- wxGridSizer

  Els objectes d'aquesta classe són molt semblants als anteriors, però aquests divideixen l'objecte horitzontalment i verticalment alhora, a més de mantenir les divisions creades alineades.

- wxSpacer

  Els objectes d'aquesta classe s'utilitzen per generar espais entre dos elements tant horitzontalment com verticalment

- wxPanel

  Els objectes d'aquesta classe s'utilitzen per agrupar elements dins d'un frame.

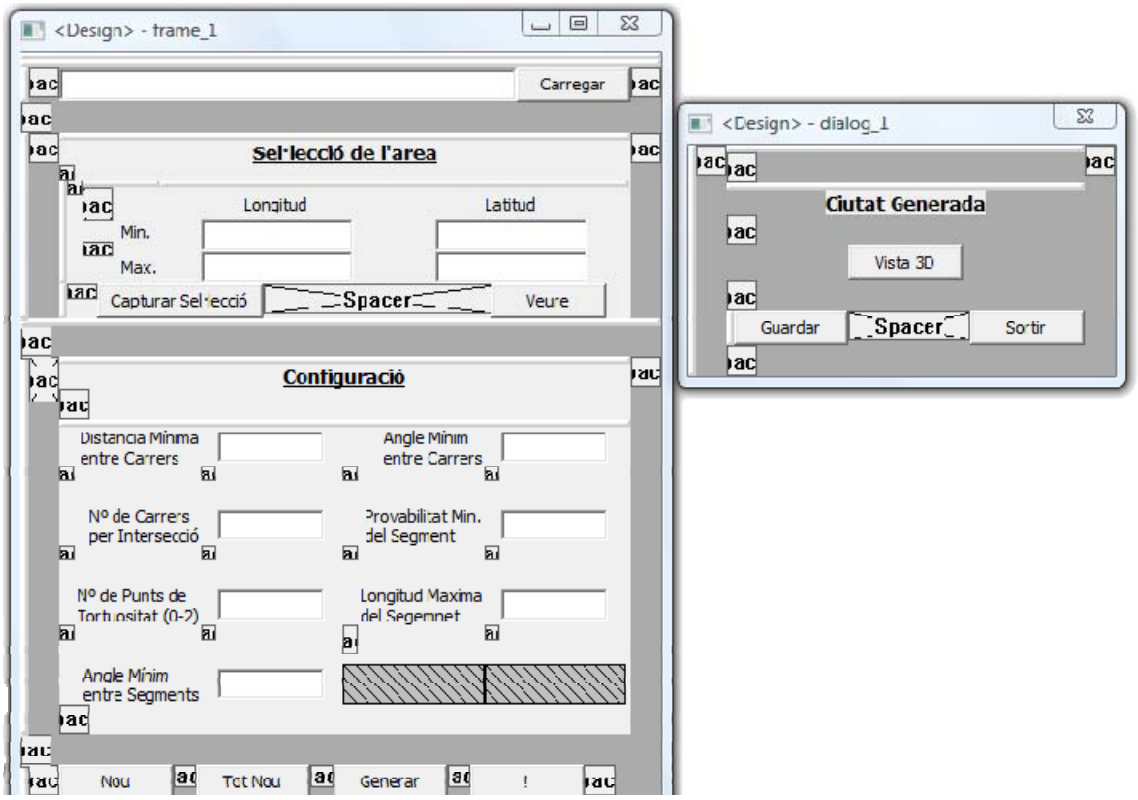


Figura 70: Disseny de les dos finestres d'aquesta aplicació

Aquesta UI s'executa amb un fil d'execució propi, això s'ha fet així ja que d'aquesta manera l'aplicació resulta molt més estable que no pas si es fes servir el mateix fil d'execució que el Houdini.

6.1.1 Integració de la UI dins del Houdini

Per utilitzar la UI dins del Houdini és poden seguir un d'aquest dos passos:

- Crida des de la consola Python

  Aquesta és el mètode més senzill per fer-ho. L'únic que s'ha de fer és obrir la consola python del Houdini (s'explica com fer-ho a l'apartat 2.4.3.1) i escrivint "execfile('ruta on es troba l'EBStreetGenerator/EBStreetGenerator_UI.py')". El problema que té aquest mètode es que s'ha d'escriure aquesta línia cada cop que es vol executar la UI. El següent mètode soluciona aquesta problemàtica.

- Creació d'una Tool

  Aquest mètode és bastant més elegant i serà el que s'utilitzarà normalment. Primer s'ha d'escollir si es vol afegir la Tool en una de les shelfs que es tenen creades o se'n vol crea una de nova.

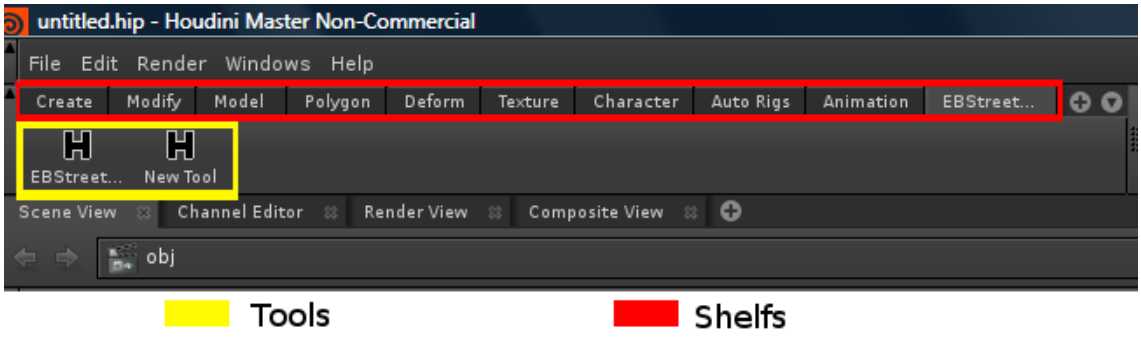


Figura 71 : Disseny Posició de les Shelf i les Tools al menú del Houdini

A continuació s'explicaran els passos per poder crear una nova shelf i dins d'aquesta una nova Tool. En el cas de voler afegir la Tool en una shelf existents anar directament al pas 5.

1. Es fa clic al símbol + que hi ha al costat de les Shelfs.
2. Es fa clic a New Shelf
3. Al diàleg que s'obre es canvien Name i Label pels noms desitjats
4. Es fa clic a Apply i després Accept i la nova Shelf ja apareix a la llista.

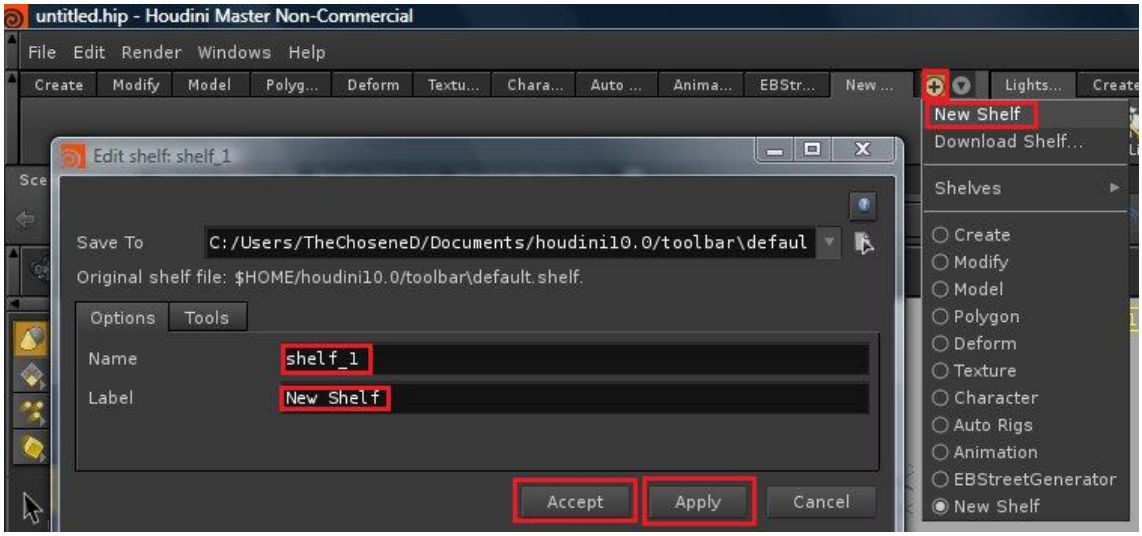


Figura 72: Passos per la creació d'una Shelf

5. Es fa clic a la Shelf on volem afegir la Tool
6. Es fa clic amb el botó dret a qualsevol lloc dins de la shelf.
7. Es clicka New Tool.
8. Dins la pestanya Script s'escriu execfile('ruta on es troba l'EBStreetGenerator/EBStreetGenerator_UI.py')
9. Es clicka Apply i després Accept i la nova Tool ja apareix a la llista.

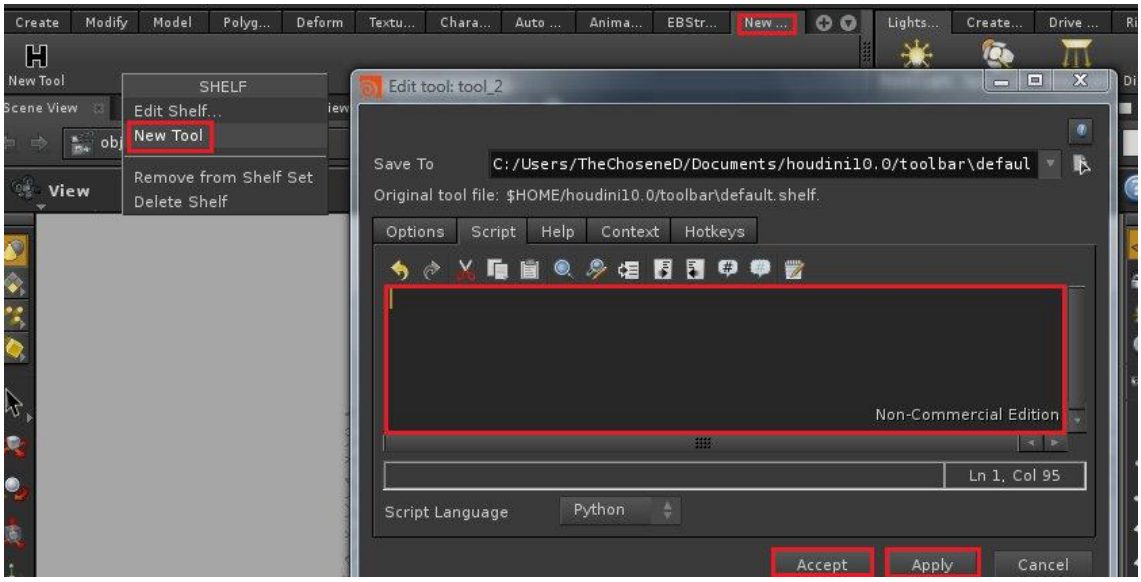


Figura 73: Passos per la creació d'una Tool

6.1.2 Utilització de la UI

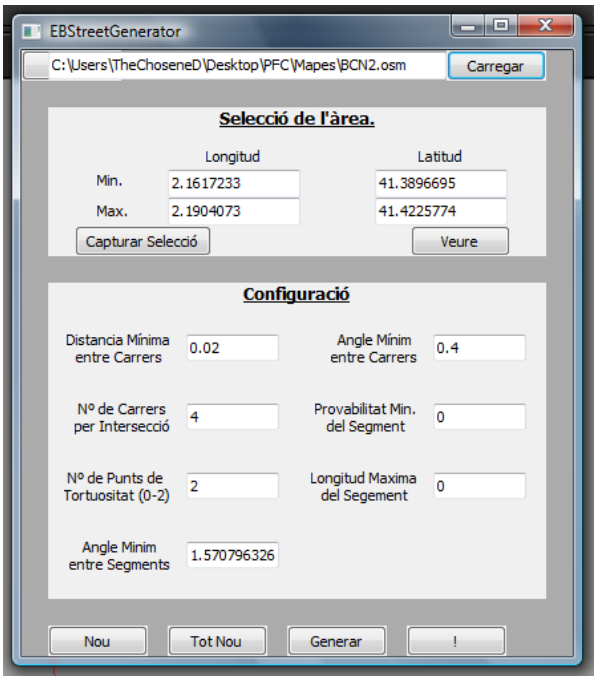


Figura 74: Finestra Principal

Per carregar un mapa OSM només s'ha de fer clic al botó "Carregar" i navegar per la finestra que s'obre fins a trobar el mapa que es desitja carregar. Si el mapa que s'ha carregat no es el que es volia, es pot tornar a repetir el mateix procediment per carregar-ne un altre, o es pot fer clic al botó "Tot Nou" per deixar la UI igual que quan s'ha obert.

Per seleccionar els punts que es desitgen utilitzar per crear la nova xarxa de carrers, es poden seguir dos camins.

1. Entrar directament les dades de les coordenades límit dels punts que es volen utilitzar, en els quadres de text de la secció selecció de l'àrea.

2. També es pot seleccionar els punts mitjançant la vista en 2D, que es genera sempre que es carrega un mapa. Només cal canviar la mida o moure el rectangle boxResize fins a que es tinguin seleccionats els punts desitjats. Un cop seleccionats es fa clic al botó "Capturar Selecció" de la UI: al fer això es llegeixen les dades del boxResize i s'escriuen als quadres de text de la secció selecció de l'àrea.

Si un cop s'han seleccionat els punts es vol veure tros de mapa que formen només s'ha de fer clic a "Veure". Per tornar a la selecció inicial de tot el mapa només s'ha de fer clic a Nou.

Un cop s'han seleccionat els punts ja es pot fer clic a "Generar" per tal de generar la nova xarxa de carrers. Aquest procés tarda més o menys a realitzar-se depenent del nombre de punts seleccionats. Un cop s'hagi acabat aquest procés apareixerà una nova finestra amb tres opcions.

1. **Generar Ciutat**: Genera la vista 3D de la xarxa de carrers generada

2. **Guardar**: S'obre una nova finestra on es pot seleccionar el nom i la destinació on es guardarà un arxiu OSM amb la xarxa de carrers creada.

3. **Sortir**: Retorna a la finestra principal de l'aplicació.

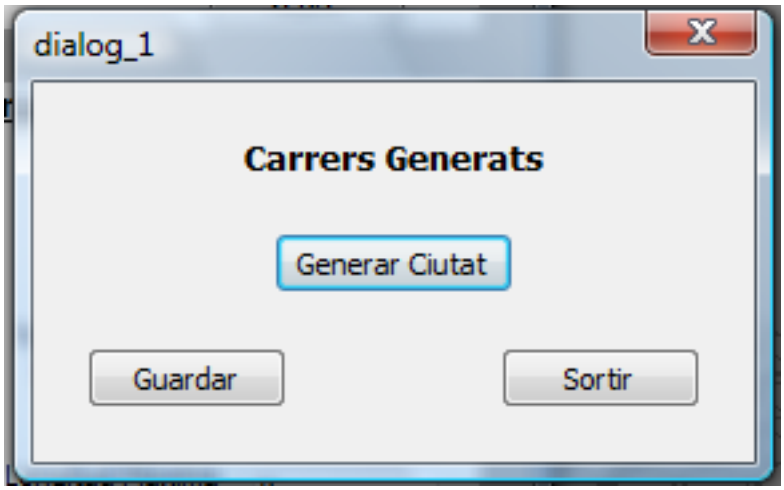


Figura 75: Finestra que apareix un cop s'ha finalitzat la creació dels carrers.

6.2 Càrrega del fitxer OSM

El primer pas a seguir és llegir el fitxer OSM escollit per l'usuari i carregar-lo a la memòria per tal de poder-lo com a exemple per generar una nova xarxa de carrers. Per això es fa servir el mòdul minidom de Python, del que ja s'ha parlat a l'apartat 2.4.2.

Es fan servir les següents sentències:

```
osmMap=open("map.osm","r")
root=minidom.parseString(self.clean(osmMap))
```

Un cop executat, la variable root conté un arbre de nodes amb totes les dades del fitxer "map.osm", veure figura 7. Aquestes sentències es troben dins el mètode __init__ de la classe osmTree.

6.3 Generació del fitxer OSM

Un cop generada la xarxa de carrers, aquesta es pot guardar en un fitxer OSM. Per fer això s'utilitzen les següents sentències

```
Path="C:\newMap.osm"
f=codecs.open(Path,'w',encoding='utf-8')
root.writexml(f)
f.close()
```

Un cop executat, el fitxer "newMap.osm" conté les dades de l'arbre root convertides al format Osm. Aquestes sentències es troben dins el mètode CreateOSMFile de la classe osmTree.

6.4 Conversió de l'arbre miniDom a un Graf

Un dels passos més importants és la conversió l'arbre miniDom que conté les dades del mapa carregat a un graf per tal de poder-hi treballar millor. Per fer-ho s'utilitzen les sentències següents:

```
    StreetsList=osmT.getStreetsList()
    PointList=osmT.getNodeList()
1   for point in PointList:
        Graph[point.getAttribute('id')]={}

    for street in StreetsList:
        StreetPoints=getElementsByTagName('nd')
        i=0
        streetLenght=len(StreetPoints)
        After=StreetPoints[1]
        Before=None
        while i<streetLenght:
            current=StreetPoints[i]
2           edges=Graph[current.getAttribute('ref')]
            if After !=None:
                edges[After.getAttribute('ref')]=1
            if Before !=None:
                edges[Before.getAttribute('ref')]=1
            Before=current
            i+=1
            if (i+1)>=streetLenght:
                After=None
            else:
                After=StreetPoints[i+1]
```

Aquest codi es troba dividit en 2 blocs:

- En el primer es crea un nova entrada al diccionari Graph amb els identificadors de cadascun dels nodes que hi ha dins de la variable PointList.

- En el segon es fa un recorregut per a cadascun dels carrers que hi ha dins la variable StreetsList i s'afegeix un aresta de cost 1 entre dos nodes, si aquests es troben un rere l'altre dins el llistat de punts que formen el carrer.

Un cop executat, la variable Graph conté el graf de l'arbre osmT en el format indicat a l'apartat 4.3.3. Aquestes sentències es troben dins el mètode __Init__ de la classe mapGraph.

6.5 Càlcul de les estadístiques

Un cop ja es té el mapa OSM carregat en memòria, el següent és calcular les estadístiques d'aquest arbre, per fer-ho s'utilitza el mètode CalculateStreetStatistics de la classe dataHandler.

```
def CalculateStreetStatistics(self, Tree):
        streets=Tree.getStreetsList()
        Nodes=Tree.getIntersectionNodes()
    1   st=osmStatistics(Nodes)
        graph=mapGraph(Tree.getStreetsList(),Tree.getNodeList())
        Dgraph= graph.getGraph()
        coords=Tree.getPointsCoordinates()

        for street in streets:
            elements=street.childNodes
    2       st.calculateLevel(elements)
            st.calculateTortuosity(elements,coords)
            st.calculateDistanceMean(elements,coords)
            AMean=statistics.calculateAngleMean(elements,coords)
            st.calculateAngleVariance(elements,coords,AMean)

    3   st.calculateIntersectionDistanceVariance(Dgraph, coords)

        for intersec in Nodes:
            T=st.getTortuosity(intersec)
            T= sum(T)/len(T)
            DM=st.getDistanceMean(intersec)
            DM= sum(DM)/len(DM)
            DV=st.getDistanceVariance(intersec)
    4       AM=st.getAngleMean(intersec)
            AM= sum(AM)/len(AM)
            AV=st.getAngleVariance(intersec)
            AV=sum(AV)/len(AV)
            if len(T)>0 and len(DM)>0 and len(AM)>0 and len(AV)>0:
                st.setValues(intersec,T ,DM ,DV,AM ,AV)
            else:
                st.setValues(intersec,0,0,0,0,0)
        return st,coords
```

On el paràmetre Tree representa un objecte de la classe osmTree. Aquest codi es troba dividit en 4 blocs:

- En el primer bloc es defineixen les variables:
    - streets: subarbre de Tree amb tots els nodes del tipus way
    - Nodes: subarbre de Tree amb tots els nodes del tipus way.
    - st: Nou objecte de la classe statistics
    - graph: Conversió de l'arbre miniDOM Tree a un graf
    - coords: Coordenades de tots els punts de Tree

- En el segon es calculen les estadístiques de cadascun dels carrers
- En el tercer es calcula la variança de la distància
- En el quart es calcula la mitjana de les propietats tortuositat, mitjana de la distància, mitjana de l'angle i variança d'aquest, que es troben guardades a cadascuna de les interseccions. La mitjana calculada es guarda a les interseccions substituint als valors que ja hi havia.

6.6 Generació de la nova xarxa de carrers

Quan ja s'han calculat les estadístiques el següent pas és generar la xarxa de carrers, per fer-ho s'utilitza el mètode generateStreets de la classe streetGenerator.

```
def generateStreets(self):
        city=osmGraph('')
        CoordKeys= self.Coordinates.keys()
        city.addPoints(CoordKeys,self.Coordinates)
        levels=self.LevelIntersection.keys()
        levels.sort(reverse=True)
        Nstreets=[]
        for level in levels:
            LStreets=[]
            Points=self.LevelIntersection[level]
            Continue=True
            while self.avariableSlots(Points) and Continue:
                Continue=False
                for point in Points:
                    AuxS=self.randomWalk(point,Points,city)
                    if len(AuxS)>1:
                        Continue=True
                        Nstreets.append(Auxstreets)
                        city.addStreet(AuxS)
                        LStreets.append(Auxstreets)
        return city,Nstreets
```

Aquest codi crea un objecte osmGraph (city) on es guardarà la nova xarxa de carrers i crea carrers (utilitzant mètode randomWalk) unint punts del mateix nivell fins que no en pot crear més. Això es pot deure a que tots els punts han arribat al nombre màxim de connexions o a que el mètode randomWalk és incapaç de crear més carrers (les causes s'expliquen en el següent apartat.).

6.6.1 Generació d'un carrer

```
def randomWalk(self,Point,PointsList,city):
        TP=Configuration.TortuosityPoints
        PL= PointsList
        stop=False
        street=[]
        Fpoint=False
        Flenght=0
        positive=True
        if len(PointsList)>1:
            Cpoint=Point
```

```
            Ppoint=''
            street=[Cpoint]
            while not stop and self.nUsedIntersection[Cpoint]<4:
                Npoint=self.choosePoint(Cpoint,PL ,Ppoint,street)
                if Npoint=='':
                    stop=True
                else:
                    edg=[Cpoint, Npoint]
                    Plist=self.generateEdgeTortuosity(edg,positive,TP)
                    positive=not positive
                    start=Cpoint
                    for Point in Plist:
                        NewPoint=(str(Point[0]),str(Point[1]))
                        p=city.addNewPoint(NewPoint)
                        city.addEdge(start, p)
                        street.append(p)
                        start=p
                    street.append(Npoint)
                    city.addEdge(start, Npoint)
                    self.nUsedIntersection[Cpoint]+=1
                    self.nUsedIntersection[Npoint]+=1
                    Ppoint=Cpoint
                    Cpoint=Npoint
        return street
```

Aquest codi busca un punt dels del paràmetre PointsList, per poder-lo enganxar amb Point, i aquest punt poder-lo enganxar amb un altre, això es va repetint fins que s'arriba a un punt en que no se'n poden enganxar més. Per seleccionar el punt següent s'utilitza el mètode choosePoint.
Un cop s'ha escollit el punt següent, es calculen i s'agregen al graf (city) el punts de tortuositat i les arestes per unir aquests amb el punt escollit i Point. Finalment s'incrementa el contador d'usos d'aquest punt i de Point (usos guardats a nUsedIntersection).

6.6.2 Elecció d'un punt

```
def choosePoint(self,Cpoint,PointsList,Ppoint,street):
        SelectablePoints=[]
        sPoints=[]
        Npoint='''
        MSL= Configuration.MaxSegmentLenght
        MSI=Configuration.MaxSegmentsIntersection
        MSA=Configuration.MaxSegmentAngle
        MSP= Configuration.MinSegmentProvability

        for Point in PointsList:
            creua=self.crossEdge(Cpoint,Point)
            CpLon= float(self.Coordinates[Cpoint][0])
            CpLat=  float(self.Coordinates[Cpoint][1])
            PLon= float(self.Coordinates[Point][0])
            PLat=  float(self.Coordinates[Point][1])
            distance=geo.distance(CpLon, CLat,PLon,PLat)

            if Ppoint=='':
                angle=self.Statistics[Cpoint][3]
            else:
               PC=self.Coordinates[Point]
               PPC= self.Coordinates[Ppoint]
               angle=geo.angle(self.Coordinates[Cpoint],PPC ,PC)
```

```
            if angle<0:
                angle=2*math.pi+angle

            goodangle= angle>MSA and angle<((math.pi*2)-.MSA)
            if MSL==0 or distance<MSL) :
                used=self.nUsedIntersection[Point]
                tS= self.tooSimilar(Cpoint,Point)
                cP= self.crossPoint(Cpoint,Point)
                Dif= Point!=Cpoint
                GoodEdge= not creua and not tS and not cP
                if Dif and used< MSI and GoodEdge and goodangle:
                    if Ppoint!='':
                        angle=geo.angle((CpLon,CpLat),PPC,(PLon,PLat))
                    else:
                        angle=self.Statistics[Cpoint][3]

                    DM= self.Statistics[Cpoint][1]
                    AM= self.Statistics[Cpoint][3]
                    DV= self.Statistics[Cpoint][2]
                    AV= self.Statistics[Cpoint][4]
                    D=distance
                    Prob=BiGauss.probability(D,angle,DM,AM,DV,AV)
                    PSP= not (Point in sPoints)
                    if Prob>MSP and PSP and not(Point in street):
                        SelectablePoints.append([Prob,Point])
                        sPoints.append(Point)
        Npoint=CDF.cdf(SelectablePoints)
        return Npoint
```

Aquest codi escull els punts en que l'aresta per unir-los amb Cpoint compleixin:

- no en creui cap de les existents
- no tingui una longitud major que una distància definida per l'usuari (Configuation.MaxSegmentLenght)
- no passi a una distància menor a una distància definida per l'usuari (Configuation. MinStreetsDistance) de qualsevol dels punts possibles.
- No tingui un angle inferior a un angle definit per l'usuari(Configuation. MinStreetsAngle) respecte a cap de les arestes existents que passin per ell ni per Cpoint.
- Tingui un angle respecte a l'anterior aresta del carrer entre el valor d'un angle definit per l'usuari (Configuation.MaxSegmentAngle) i el mateix angle en sentit negatiu

Un cop escollits es calcula la probabilitat de cadascun d'ells utilitzant el mètode probability de la classe BiGauss, aquesta probabilitat es guarda en una llista com a un parell id/probabilitat.

Finalment s'escull un punt utilitzant el mètode cdf de la classe CDF.

6.7 Generació de la imatge 2D d'una xarxa de carrers

Per poder dibuixar una xarxa de carrers en 2D al Houdini, s'utilitzen les següents sentències.

```
streets,ID=DataHandler.streets(graph)
CleanScreen()
container =
hou.node("/obj").createNode("geo")
houdiniOSM.streets(container, streets)
```

El primer que es fa és obtenir les coordenades dels punts que formen cadascun del carrers i es guarda a la variable streets. Seguidament es neteja la pantalla del Houdini (per si ja hi havia alguna cosa dibuixada) i es crea un node contenidor Houdini de tipus geo que es guarda a la variable container.
Finalment es crida el mètode streets de la classe houdiniOSM per tal de crear tota la geometria dels carrers i mostra-la per la pantalla del Houdini.

6.8 Generació de la imatge 3D d'una xarxa de carrers

Per poder dibuixar una ciutat 3D, a partir d'una xarxa de carrers, al Houdini. S'utilitzen les següents sentències.

```
blocks,ID=DataHandler.blocks(map.graph)
CleanScreen()
XML=EBStreetGeneratorPath + 'uE_EBStreetGeneratorConfig.xml'
readInput(urbanEngineDemosPath,XML)
uE = uE_EBStreetGeneratorFactory(blocks)
network.createGeometry(uE)
```

El primer que es fa és obtenir les coordenades dels punts que formen cadascuna de les illes i es guarda a la variable blocks. Seguidament es neteja la pantalla del Houdini (per si ja hi havia alguna cosa dibuixada) i es llegeix el fitxer de configuració de l'urbanEngine. A continuació es crea un objecte, anomenat uE, de la classe uE_EBStreetGeneratorFactory
Finalment es crida el mètode createGeometry de la classe network per tal de crear tota la geometria necessària per mostrar la ciutat en 3D per la pantalla del Houdini.

6.9 Exemple Main

A continuació es mostren les sentències necessàries per obrir un mapa OSM, generar una xarxa de carrers semblant a la del mapa i guardar la xarxa de carrers generada en un arxiu OSM.

```
path="c:\ "
test=osmTree(path+"mapa.osm")
test.eraseNoTypeElements(u'highway')
test.deleteNodeMarks()
st,coord=DataHandler.CalculateStreetStatistics(test)
city=streetGenerator(st.Intersections,coord,st.Level)
graph,streets=city.generateStreets()
osm=graph.GenerateOSMTree()
osm.CreateOSMFile(path)
```

El primer que es fa és convertir el fitxer en un objecte de la classe osmTree i es guarda a la variable test. Seguidament s'eliminen d'aquest objecte tots els elements que no

siguin carrers ni nodes. Un cop l'arbre test és net, se'n calculen les estadístiques i es guarden a la variable st utilitzant el mètode CalculateStatistics de la classe DataHandler. Un cop les estadístiques es troben a la variable st es crea un objecte de la classe streetGenerator anomenat city. L'objecte city, un cop creat, genera la nova xarxa de carrers cridant al mètode generateStreets i guarda aquesta nova xarxa a graph.
Per finalitzar es converteix el graf graph en un objecte de la classe osmTree anomenat osm i aquest s'utilitza per crear el fitxer OSM amb la xarxa de carrers creada anteriorment.

7-Resultats

Dins d'aquest capítol es mostren tres exemples dels resultats obtinguts amb el projecte. De cadascun d'aquests exemples és mapa el mapa utilitzat com a exemple, el mapa de la xarxa de carrers generada i la vista en 3D d'aquesta xarxa.

7.1 Exemple mapa regular (Boston)

A continuació es mostraran les imatges corresponents a un exemple del resultat d'aquesta aplicació al utilitzar un mapa amb una distribució regular. La figura 76 mostra el mapa original, en aquest cas es tracta d'un trosset de Boston. A continuació en la figura 77, es pot veure el mapa resultat, un cop s'ha processat l'original. Finalment, a la figura 78, es mostra la vista en 3D de la xarxa de carrers generada.

Figura 76: Mapa OSM d'un districte de Boston

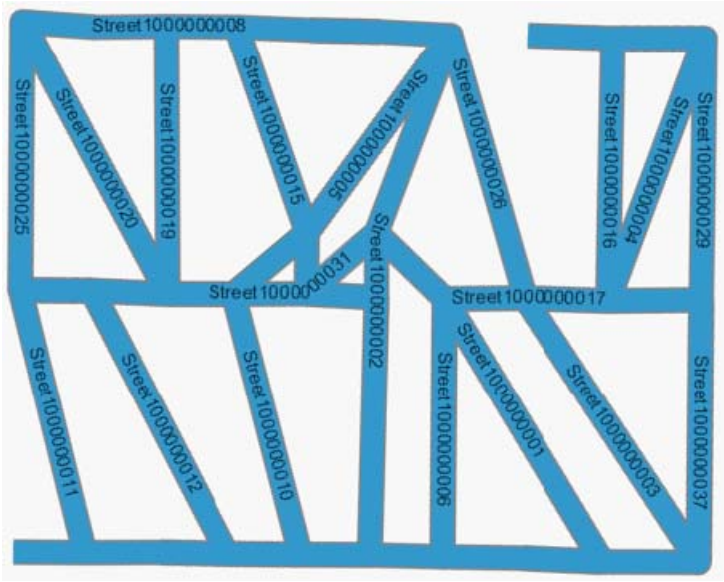


Figura 77: Mapa resultant d'utilitzar el districte de boston anterior.

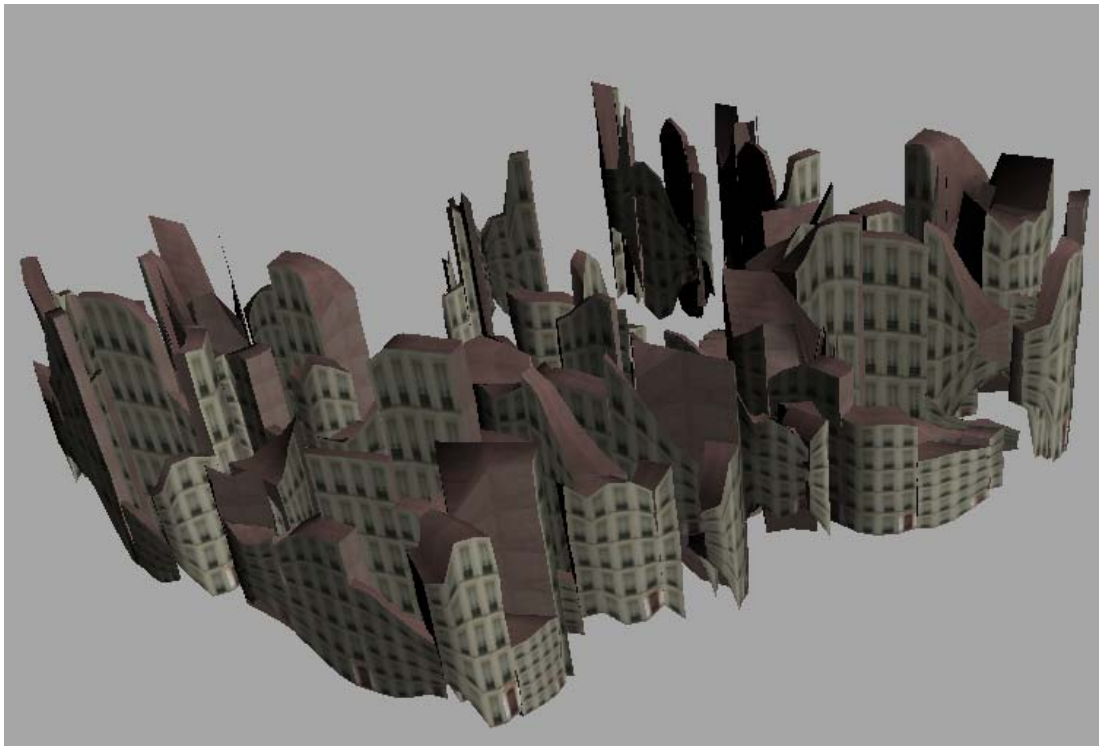


Figura 78: Vista en 3D de la xarxa de carrers de la figura 77

7.2 Exemple mapa irregular (Sidney)

A continuació es mostraran les imatges corresponents a un exemple d'aquesta aplicació al utilitzar un mapa amb una distribució irregular. La figura 79 mostra el mapa original, en aquest cas es tracta d'un trosset de Sidney. A continuació en la figura 80, es pot veure el mapa resultat. Un cop s'ha processat l'original, i finalment a la figura 81 es mostra la vista en 3D de la xarxa de carrers generada.

Figura 79: Mapa OSM d'un districte de Sidney

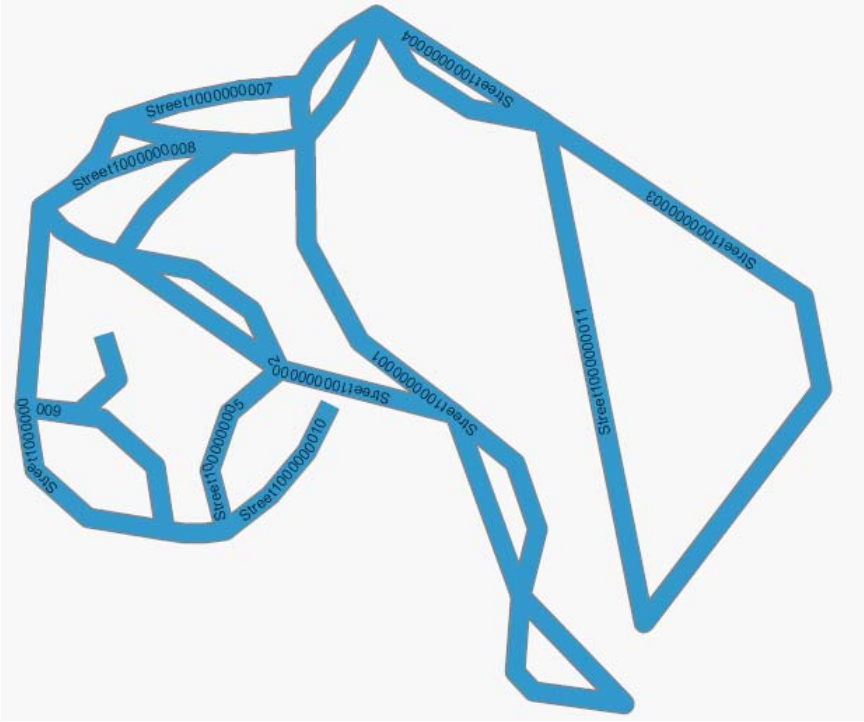


Figura 80: Mapa resultant d'utilitzar el districte de Sindney anterior.

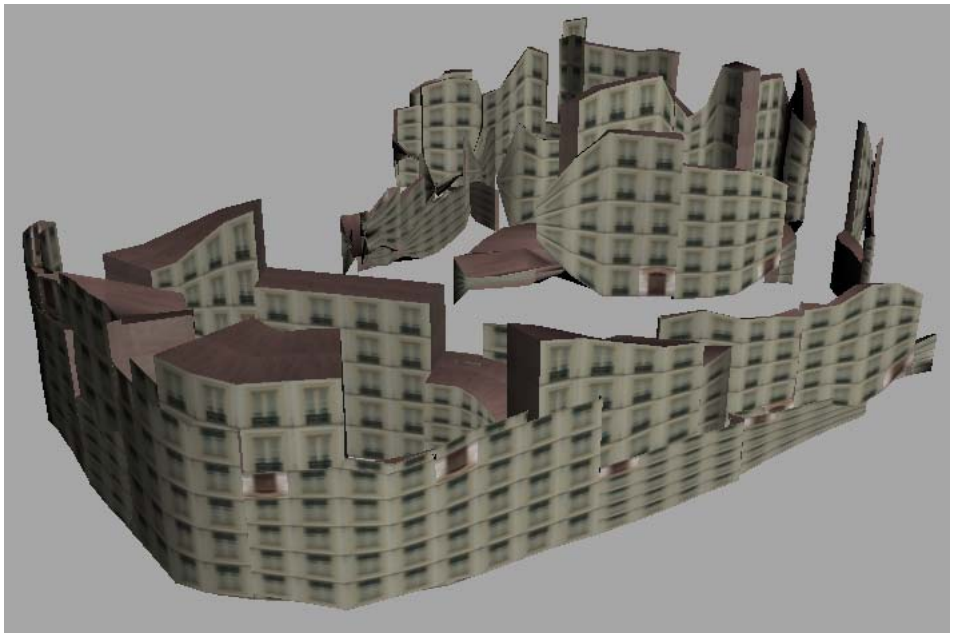


Figura 81: Vista en 3D de la xarxa de carrers de la figura 80

7.3 Exemple mapa mixte (Paris)

A continuació es mostraran les imatges corresponents a un exemple al utilitzar un mapa amb un troç en distribució irregular i un altre de regular. La figura 82 mostra el mapa original, en aquest cas es tracta d'un trosset de Paris. A continuació, en la figura 83 es pot veure el mapa resultat, un cop s'ha processat l'original, i finalment, a la figura 84, es mostra la vista en 3D de la xarxa de carrers generada.

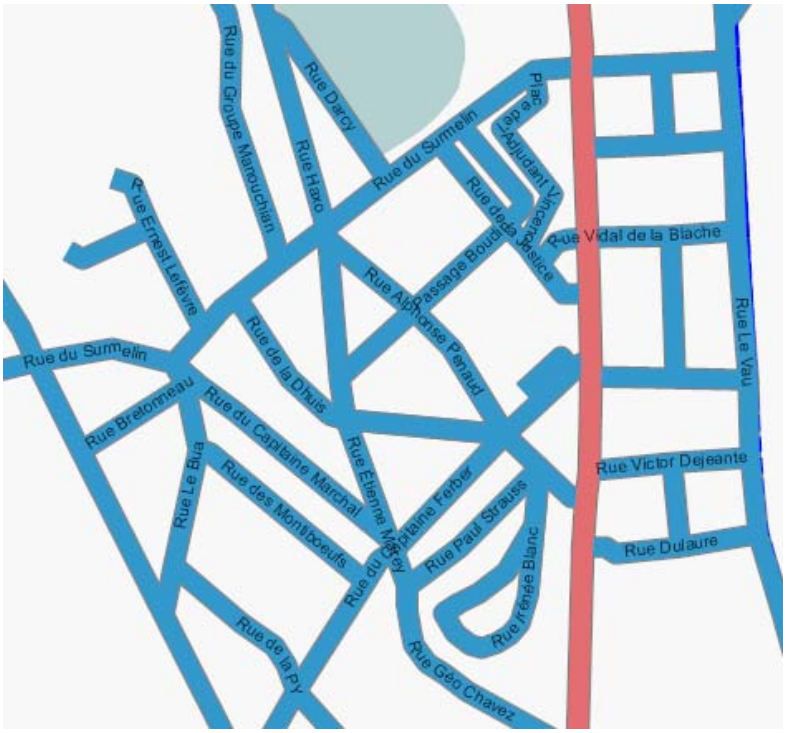


Figura 82: Mapa OSM d'un districte de Paris

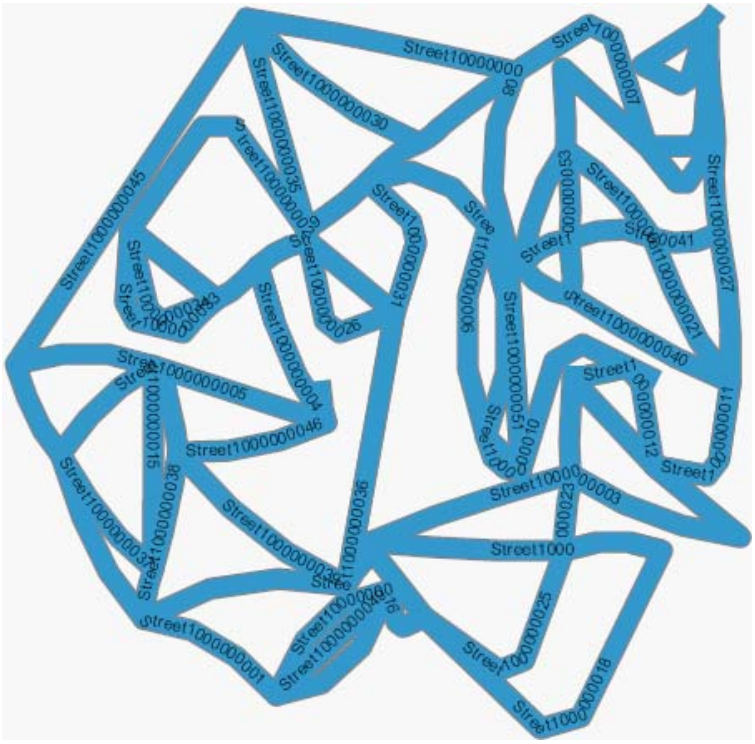


Figura 83: : Mapa resultant d'utilitzar el districte de Paris anterior

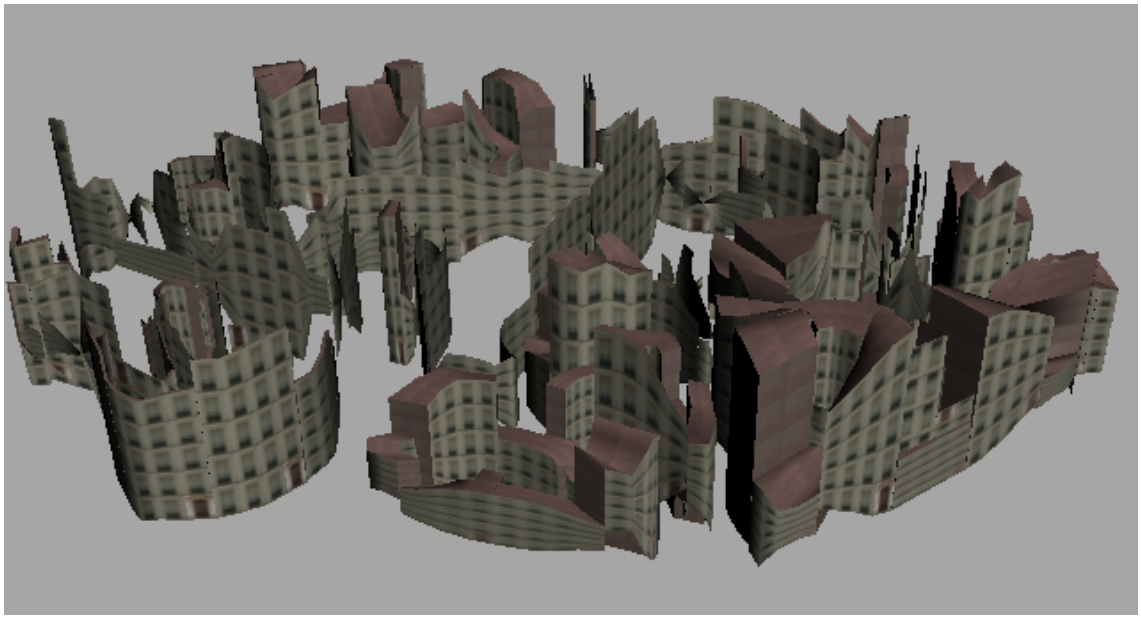


Figura 84: Vista en 3D de la xarxa de carrers de la figura 83

7.4 Exemple (Barcelona)

A continuació es mostraran les imatges corresponents a un altre exemple del resultat d'aquesta aplicació. La figura 85 mostra el mapa original, en aquest cas es tracta d'un trosset de Barcelona. A continuació en la figura 86, es pot veure el mapa resultat, un cop s'ha processat l'original, i finalment, a la figura 87, es mostra la vista en 3D de la xarxa de carrers generada.

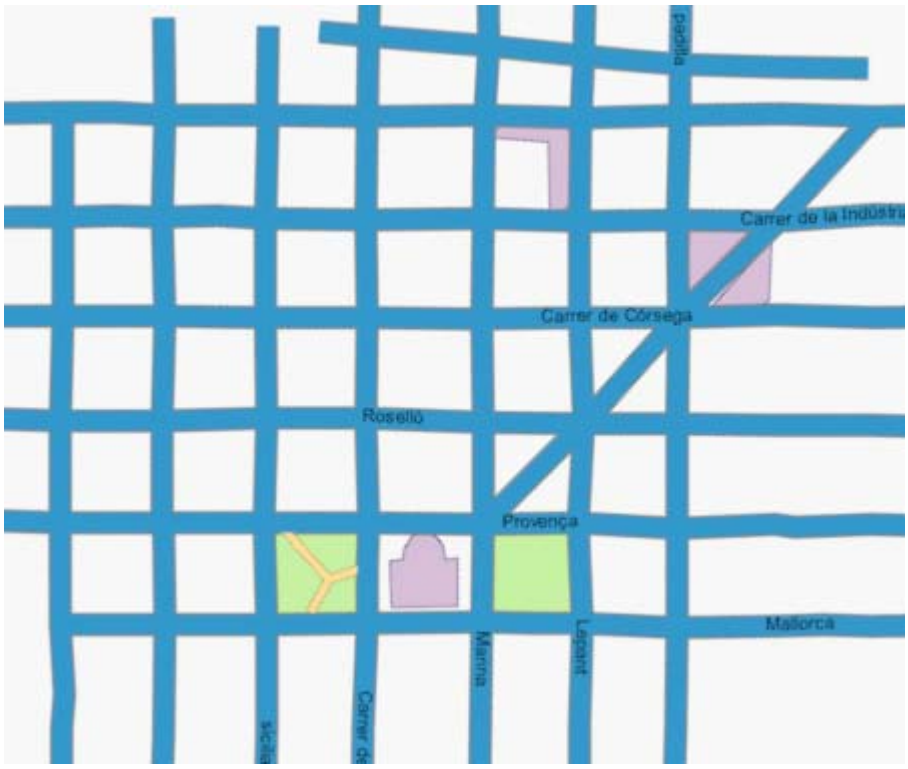


Figura 85: Mapa OSM d'un districte de Barcelona

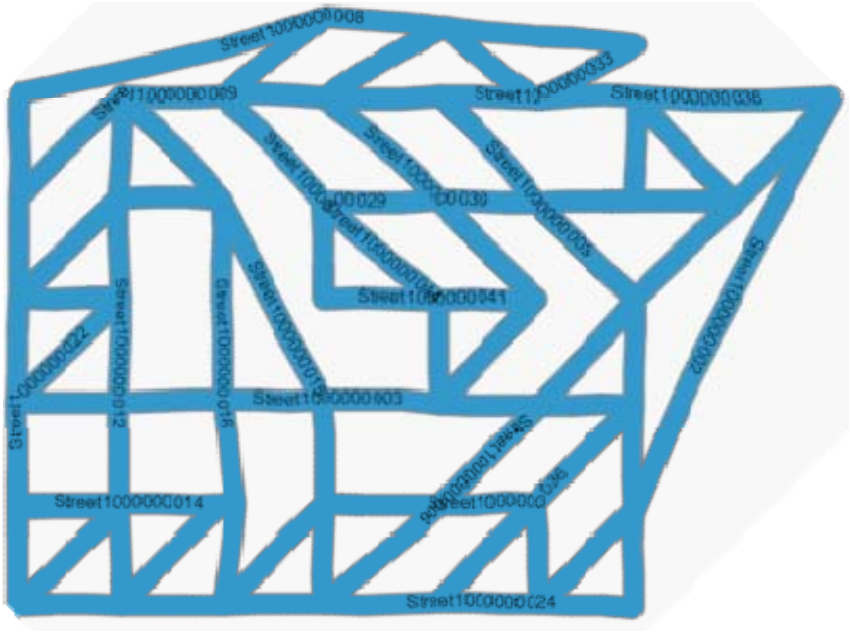


Figura 86: : Mapa resultant d'utilitzar el districte de Barcelona anterior

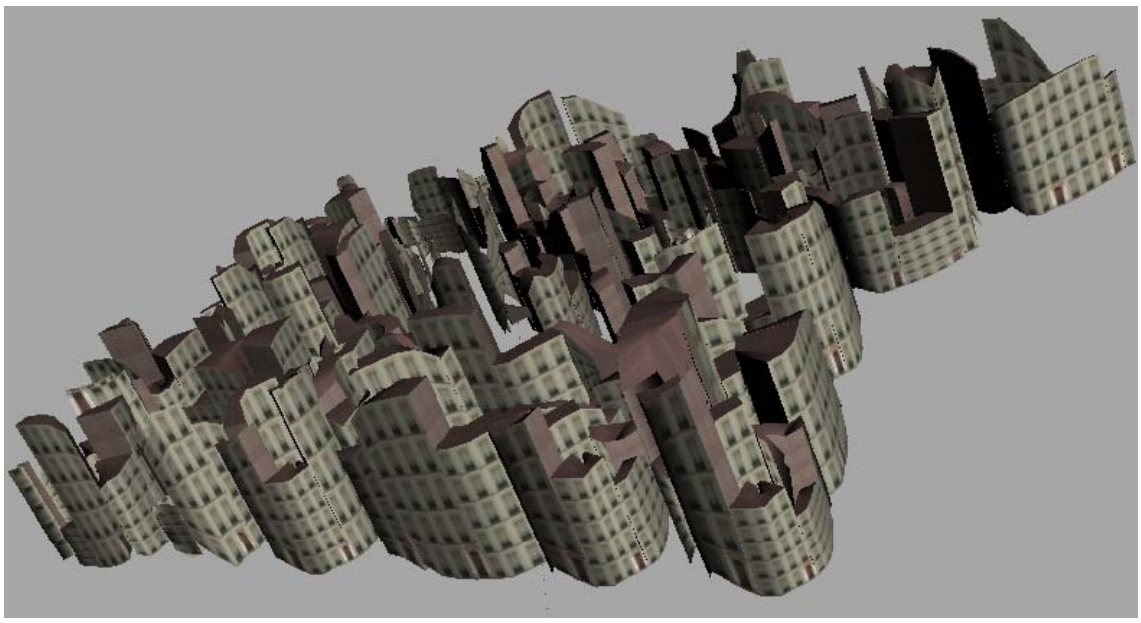


Figura 87: Vista en 3D de la xarxa de carrers de la figura 85

8-Conclusions

L'objectiu principal d'aquest projecte consistia en crear un programari que permetés enerar xarxes de carrers a partir d'exemples. Aquest objectiu ha estat complert de forma satisfactòria així com tot el conjunt de requeriments definits des d'un principi.

En un principi, el camí a seguir era el marcat per l'article "Interactive Example-Based Urban Layout Synthesis" (D. Aliaga, C. Vanegas i B. Benes, 2008). S'ha aconseguit un sistema per a crear xarxes de carrers basades en exemples bastant semblant al descrit per l'article, tot i que s'han hagut de solventar algunes problemàtiques no explicades completament a l'article.

A més també s'ha aconseguit crear un conjunt de classes genèriques per a treballar amb mapes OSM que poden ser utilitzades per a futures aplicacions que treballin amb aquest format.

El desenvolupament d'aquest projecte ha permès aconseguir dos típus d'objectius. Per una banda els relacionats amb els requeriments de l'aplicació i per l'altre els referents a l'aprenentatge de temes que tenen a veure amb la geometria, l'estadistica i dels diferents softwares utilitzats.

Pel que fa a l'aprenentage s'ha aconseguit ampliar els coneixaments relacionats amb els conceptes següents:

- Algorismes de càlcul de relació entre vectors, segments i punts.
- Les funcions utilitzades en estadística: "Comulative Distribution Function" i la funció Gaussiana.
- Eines de software com ara wxPython, miniDom, urbanEngine, Houdini...
- El llenguatge de programació Python

9-Treball de Futur

A partir del treball fet en aquest projecte es podrien implementar un gran nombre de millores i ampliacions.

Una millora que s'hi podria fer seria afegir suport per a més formats de mapes vectorials, una altre podria ser fer que a partir de diferents mapes d'exemple, l'usuari els pogués unir per a generar una ciutat més gran.

Les classes creades per treballar amb mapes OSM es podrien fer servir per crear programes que, per exemple, calculin el camí mínim entre dos punts d'un mapa o que calculin quina és la menor ruta per visitar tota una seria de punts prèviament marcats al mapa...

10-Agraïments

M’agradaria, a través d’aquest últim capítol, donar les gràcies a totes aquelles persones que m’han ajudat durant el transcurs d’aquest Projecte Final de Carrera.

- En Carlos Venegas, per l’ajuda que m’ha proporcionat a l’hora d’entendre algun dels mètodes descrits a l’article “Example-Based Urban Layout Synthesis”

- Al meu Tutor en Gustavo Patow, per la seva gran ajuda i per saber posar-me pressió quan calia.

- Finalment també m’agradaria agrair el suport que he rebut per part de la meva família i amics.

11-Bibliografia

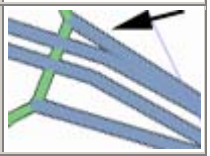
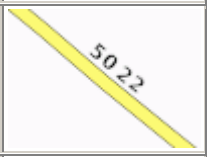
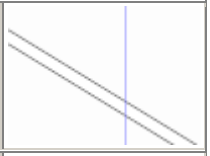
1. D. Aliaga, C. Vanegas i B. Benes (2008). Example-Based Urban Layout Synthesis.
2. Tutorial Python: http://docs.python.org/3.0/tutorial/
3. Format OpenStreetMap: http://wiki.openstreetmap.org/wiki/Develop
4. Tutorial XML DOM: http://wiki.python.org/moin/MiniDom
5. World Wide Web Consortium: http://www.w3.org/
6. Relacions entre figures Geomètriques:
    - http://local.wasp.uwa.edu.au/~pbourke/geometry/
    - http://www.euclideanspace.com/
7. Funcions matemàtiques: http://mathworld.wolfram.com/
8. Documentació Houdini: http://www.sidefx.com/docs/houdini10.0/
9. Tutorial wxPython amb Houdini: http://pythonandhoudini.simonpate.co.uk/

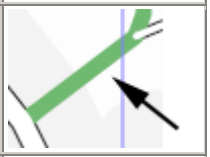
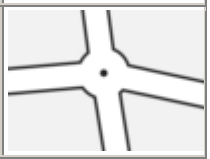
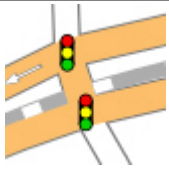
ANNEX 1

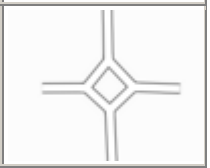
Característiques físiques

o Carreteres (Highway)

Par a elements lineals (vies), l'etiqueta "highway" pren un dels següents valors:

Valor	Element	Comentari	Exemple
motorway		Autopistes y Autovies,	
motorway_Linke		Enllaços a autopistes y autovies	
trunk		Carreteres nacionals (en trams que no siguin autopista/autovia).	
trunk_Linke		Rampes d'excés a carreteres nacionals.	
primary		Carreteres autonòmiques de primer nivell.	
primary_Linke		Vies que connecten carreteres secundaries amb qualsevol altre tipus de Via.	
secondary		Carreteres autonòmiques de segon nivell.	
tertiary		Carreteres autonòmiques de tercer nivell.	
unclassified		Altres carreteres, sense identificació pròpia.	
track		Camins forestals, camins sense asfaltar.	

residential		Carrers dins del nucli urbà, se n'exclouen les travessies.	
living_street		Carrers residencials on la prioritat la tenen els vianants.	
service		Vies de servei; accessos a pàrkings, gasolineres...; Qualsevol via que la seva finalitat nio sigui la de transit de vehicles continuat.	
bridleway		Camins on es permet el pas de vehicles de tracció animal.	
cycleway		"Carril-bici".	
footway		Vies para l'ús exclusiu de vianants, que no son carrers peatonals.	
pedestrian		Carrers peatonals (inclús si es permet el pas de vehicles d0emergencia i servei.	
steps		Trams d'escales (en vies per a l'és dels vianants)	
mini_roundabout		Mini-rotonda: Una rotonda que només consta d'una marca vial al terra.	
stop		Senyal de stop.	
traffic_signals		Semàfor.	
crossing		Pas de vianants	
traffic calming		Elements per reduir la velocitat: ressalts, bades reductores de velocitat, etc	

gate		Portal (par a l'ús de vehicles).	
stile		Pas en un mur o balla, per a l'ús dels vianants.	
cattle_grid		Barrera Canadenca. Reixa al terra que permet el pas de vehicles i vianants però no del bestiar.	
toll_booth		Cabina de peatge	
incline		Pendent, rampa.	
incline_steep		Pendent pronunciada, amb possibilitat de trobar vehicles lents.	
viaduct		Viaducte. Pont alt i/o llarg	
motorway_junction		Incorporació o sortida d' autopista.	
services		Àrees de servei	
ford		Bado.	
bus_stop		Parada d'autobús	
roundabout		Rotondes o glorietes.	
construction		Per a carreteres en construcció, utilitzar amb *construction=**	
Definit per l'usuari		Per a qualsevol objecte lineal de carreteres, por exemple *construction=motorway*. Utilitzar amb *highway=construction*	

o Barreres (Barriers)

Valor	Element	Comentaris	Renderització	Foto
Linear barrier				
hedge		Arbust		
fence		Balla		

wall		Mur		
ditch		Un rec o barranc, amb un corrent d'aigua, que no és fàcil de creuar		
Linear barrier with sides				
retaining_wall		Mur de contenció		
city_wall		Muralla		
Node barrier				
bollard		Pilones		
cycle_barrier		Barreres per impedir la circulació de bicicletes.		
cattle_grid		Un tipus de "porta" per no deixar passar el bestiar.		
toll_booth		Cabina de peatge		
Access node				
entrance		Forat en una barrera lineal sense cap tipus de construcció per limitar-ne el pas.		
gate		Porta		

stile		Escalons que permeten passar per sobre d'una balla		
sally_port		Portalada per travessar grans murs o muralles.		

o Vies ciclistes (Cycleway)

Valor	Element	Comentari	Exemple
lane		Carril (incorporat a una carretera)	
track		Pista (separada de la carretera).	
opposite_lane		Carril en sentit contrari al de la circulació	
opposite_track		Pista en sentit contrario al de la circulació (separada de la carretera)	
Definit per l'usuari			

o Pista (Tracktype)

Valor	Element	Comentari	Renderització	Exemple
grade1		Pista pavimentada o de base molt compacte.		
grade2		Pista no pavimentada; superfície de grava, terra o sorra compactada.		
grade3		Pista no pavimentada; Mitjanament compacte.		
grade4		Pista no pavimentada; marques de rodes amb vegetació entre elles		

grade5		Pista no pavimentada; lleugeres maques de rodes sense base, poc visibles respecte a l'entorn.		

o Vies d'aigua i portuàries (Waterway)

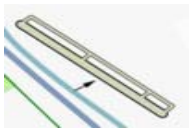
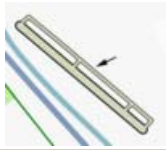
Valor	Element	Comentari	Exemple
river		Riu.	
canal		Canals, sèquies i transvasaments. Qualsevol via d'aigua artificial.	
stream		Rierol, curs d'aigua natural menor que un riu.	
drain		Drenatges y desaigües.	
dock		Moll, dàrsena.	
lock_gate		Comporta, resclosa	
turning_point		Punt on les embarcacions de major longitud que l'amplada del canal.	
aqueduct		Aqüeducte.	
boatyard		Drassana	
water_point		Presa d'aigua potable	
waste_disposal		Desaigua d'aigües residuals	
mooring		Amarrador, lloc on "s'aparquen" les barques.	
weir		Preses que no bloquegen totalment el pas de l'aigua.	
waterfall		Cascada.	
Definit per l'usuari			

o Ferrocarrils (Railway)

Valor	Element	Comentari	Exemple
station		Estació de ferrocarril.	
halt		Petita estació, pot ser sense andana, on els trens només es paren sota petició.	
viaduct		Viaducte de ferrocarril. Pont alt o llarg amb pilars entremitjos de recolzament.	

crossing		Punt habilitat perquè els vianants puguin creuar les vies.	
level_crossing		Pas a nivell (amb o sense barreres).	
rail		Vies de Ferrocarril.	
tram		Tramvies.	
light_rail		Metro lleuger. També parts del metro que es troben a l'aire lliure.	
subway		Metro.	
preserved		Ferrocarril d'ús exclusivament històric.	
disused		Ferrocarril en desus, però que es podria utilitzar.	
abandoned		Ferrocarril en desus, abandonat, no es podria utilitzar.	
narrow_gauge		Ferrocarril de via estreta.	
monorail		Ferrocarril que utilitza només un rail.	
Definit per l'usuari			

o Vies aeroportuàries (Aeroway)

Valor	Element	Comentari	Exemple
aerodrome		Aeroport.	
terminal		Terminal.	
helipad		Heliport, pista per helicòpters	
runway		Pista d'aterratge d'avions.	
taxiway		Pista de rodament: comuniquen pistes d'aterratge amb els altres edificis de l'aeroport.	
apron		Lloc on els avions es paren.	
Definit per l'usuari			

o Transport aeri (Aerialway)

Valor	Element	Comentari	Exemple
cable_car		Telefèrics.	

chair_lift		Telecadires.	
drag_lift		Telesquís, cables d'arrossegament.	

o Línies elèctriques (Power)

Valor	Element	Comentari	Exemple
tower		Torre elèctrica d'alt voltatge.	
line		Línea elèctrica.	

o Construccions (Man Made)

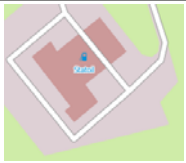
Valor	Element	Comentari	Exemple
works		Obres, fàbriques i centres de producció.	
beacon		Balisa	
survey_point		Vèrtex geodèsic, punt utilitzat en topografia.	
power_wind		Planta d'energia eòlica.	
power_hydro		Planta hidroelèctrica (incloses les de generació per onades o marees).	
power_fossil		Central elèctrica tèrmica	
power_nuclear		Central nuclear.	
tower		Torre	
water_tower		Dipòsit d'aigua elevat	
gasometer		Gasòmetre, dipòsit de gas	
reservoir_covered		Dipòsit d'aigua cobert o subterrani	
lighthouse		Far	
windmill		Moli de vent (antic)	
pier		Moll, escullera.	
Definit per l'usuari			

o Oci (Leisure)

Valor	Element	Comentari	Exemple
sports_centre		Poliesportiu	
golf_course		Camp de golf	
stadium		Estadi	
marina		Port esportiu	
track		Pista de curses (atletisme, ciclisme, hipòdrom, etc)	

pitch		Campo d'esports (camp de futbol, pista de basquet, etc)	
water_park		Parc aquàtic	
slipway		Rampa per a entrar les barques al mar	
fishing		Àrea de pesca	
nature_reserve		Parc natural, reserva ecològica	
park		Parc	
playground		Zona de jocs infantils, gronxadors,etc.	
garden		Jardí	
common		Area recreativa	
Definit per l'usuari			

o Infraestructures (Amenity)

Valor	Element	Comentari	Renderització	Exemple
pub		pub		
biergarten		Terrasses de bars i restaurants		
cafe		Cafè		
restaurant		Restaurant (no de menjar ràpid)		
fast_food		Restaurant de menjar ràpid		
parking		Aparcament.		
bicycle_parking		Aparcament per a bicicletes.		
fuel		Gasolinera		
telephone		Telèfon públic		
toilets		Urinaris públics (poden ser de pagament)		

recycling		Centres de reciclatge		
public_building		Edifici d'ús public		
place_of_worship		Centre de culte.		
grave_yard		Cementiri petit		
post_office		Oficina de correus		
post_box		Bústia de correus		
school		Escola o institut.		
university		Campus o edificis universitaris.		
college		Centres d'estudis de grau mitjà.		
pharmacy		Farmàcia	o	
hospital		Hospital		
library		Biblioteca pública		
police		Comissaria de policia		
fire_station		Parc de bombers		
bus_station		Estació d'autobusos		
theatre		Teatre		
cinema		Cinema		
arts_centre		Centro d'arts		
courthouse		Jutjats		
prison		Presó		
bank		Banc		
atm		Caixer automàtic		
townhall		Ajuntament o centre municipal		

Definit per l'usuari				

o Comerços (Shop)

Valor	Element	Comentari	Exemple
bakery		Fleca o pastisseria	
butcher		Carnisseria	
chandler		Botiga on venen espelmes i sabons	
supermarket		Supermercat o hipermercat	
Definit per l'usuari			

o Turisme (Tourism)

Valor	Element	Comentari	Exemple
information		Informació turística	
camp_site		Àrea d'acampada, càmping	
caravan_site		Àrea d'acampada per a caravanes	
picnic_site		Zona de Picnics	
viewpoint		Mirador	
theme_park		Parc d'atraccions, parc temàtic	
hotel		Hotel	
motel		Motel	
guest_house		Pensió	
hostel		Hostal, alberg	
attraction		Atracció turística	
Definit per l'usuari			

o Històric (Historic)

Valor	Element	Comentari	Exemple
castle		Castell	
monument		Monument o estàtua commemorativa gran	
museum		Museu	

archaeological_site		Jaciment arqueològic	
icon			
ruins		Ruïnes històriques	
Definit por l'usuari			

o Ús del territori (Landuse)

Valor	Element	Comentari	Exemple
farm		Terres de cultiu, granja (també per a la cria d'animals)	
quarry		Cantera, mina a cel obert	
landfill		Abocador d'escombraries	
basin		Conca hidrològica	
reservoir		Dipòsit d'aigua (potable), embassament	
forest		Zona forestal (terreny d'arbres plantats per a la seva explotació comercial).	
allotments		Horts	
residential		Us residencial, vivendes	
retail		Us comercial: principalment tendes	
commercial		Us comercial: principalment oficines	
industrial		Us industrial: fàbriques, tallers y magatzems.	
brownfield		Zona en construcció, on s'han enderrocat edificis per construir-ne de nous.	
greenfield		Zona de nova construcció	
cemetery		Cementiri.	
village_green		Zona verda pública	
recreation_ground		Zona esportiva	
Definit per l'usuari			

o Militar (Military)

Valor	Element	Comentari	Exemple
airfield		Aeròdrom militar, base aèria	
bunker		Bunker	
barracks		barracons	

danger_area		Zona de perill, àrea demarcada al voltant d'una zona d'exclusió.	
range		Camp de tir.	
Definit per l'usuari			

o Natural (Natural)

Valor	Element	Comentari	Exemple
spring		Fonts naturals	
peak		Cim d'una muntanya o turó	
cliff		Penya-segat	
scree		Zona amb perill d'esllavissades	
scrub		Zona sense cultivar coberta amb arbustos o arbres petits.	
fell		Prat.	
heath		Zona sense cultivar plena d'arbustos i amb cap o pocs arbres.	
wood		Bosc natural	
marsh		Àrea humida, maresma	
water		Aigua (llacs, llacunes, etc.)	
coastline		Línia de la costa	
mud		Fang,pantà	
beach		Platja	
bay		Badia, golf	
land		Terra existent dins d'una altre area, com per exemple un llac.	
Definit per l'usuari			

Característiques no físiques

o Ruta (Route)

Valor	Element	Comentari	Exemple
bus		Línea d'autobusos	
ferry		Ferri	
flight		Ruta aèria	
subsea		Submarina	
ski		Pista d'esquí	
tour		Visita guiada	
pub_crawl		Ruta de visita (y us) de pubs	

Definit per l'usuari			

o Fronteres, Límits (Boundary)

Valor	Element	Comentari	Exemple
national		País	
administrative		p.e. Comunitat autònoma, província, municipi...	
civil			
political		Circumscripció electoral	
national_park		Parc nacional	

o Esports (Sport)

Valor	Element	Comentari	Exemple
10pin		Bolera americana	
athletics		Atletisme	
baseball		Beisbol	
basketball		Bàsquet	
bowls		Bitlles	
climbing		Escalada	
cricket		Cricket	
cricket_nets		Xarxes d'entrenament de cricket	
croquet		Croquet	
cycling		Ciclisme	
dog_racing		Canòdrom	
equestrian		Hípica	
football		Futbol americà	
golf		Golf	
gymnastics		Gimnàstica	
hockey		Hockey	
horse_racing		Hipòdrom	
motor		Motor	
multi		Poliesportiu	
pelota		Pilota basca	
racquet		Esports de raqueta (excepte tenis)	
rugby		Rugby	

skating		Patinatge	
skateboard		Monopatí	
soccer		Futbol	
swimming		Natació	
skiing		Esquí	
table_tennis		Ping-pong	
tennis		Tenis	

Límits (Abutters)

Valor	Element	Comentari	Exemple
residential		predomini de vivendes	
retail		predomini de comerços	
industrial		predomini de tallers, fàbriques o magatzems	
commercial		edificis d'oficines, parcs empresarials, etc.	
mixed		mixta	

Mobiliari urbà

Clau	Valor	Element	Comentari
fenced	yes/no		Ballat (si/no)
lit	yes/no		Enllumenat públic (si/no)

Propietats

Clau	Valor	Element	Comentari
area	yes		Fa que una via tancada es dibuixi de color sòlid
bridge	yes		Pont
tunnel	yes		Túnel
cutting	yes		Trinxera (Tall al terreny per anivellar una carretera o ferrocarril)
embankment	yes		terraplè
lanes	Num		Número de carrils en cada direcció (o en la permesa) del segment
layer	de -5 fins a 5		Capa.
surface	paved/unpaved		Superfície pavimentada (paved) / no pavimentada (unpaved)
width	Num		Ample de la via/segment en metres

est_width	Num		Ample estimat de la via/segment en metros
depth	Num		Profunditat en metres
est_depth	Num		Profunditat estimada en metres
start_date	Date		Data d'assignació d'una propietat
end_date	Date		Data d'eliminació d'una propietat
mountain_pass	yes		Port o pas de muntanya
opening_hours	24/7 ; mo md hh:mm,hh:mm		Horari d'obertura

Restricciones

Clau	Valor	Element	Comentari
access	yes/private/ permissive/ unknown/no		Permís general d'accés. 'permissive' significa que el propietari/a permès l'accés però que el podria tallar. 'private' significa que l'accés està prohibit.
bicycle	yes/private/ permissive/ unknown/no		Permís d'accés per a bicicletes.
foot	yes/private/ permissive/ unknown/no		Permís d'accés per a vianants.
goods	yes/private/ permissive/ unknown/no		Permís d'accés per al transport lleuger de mercaderies.
hgv	yes/private/ permissive/ unknown/no		Permís d'accés par al transport pesat de mercaderies.
horse	yes/private/ permissive/ unknown/no		Permís d'accés per a genets.
motorcycle	yes/private/ permissive/ unknown/no		Permís d'accés per a motocicletes.
motorcar	yes/private/ permissive/ unknown/no		Permís d'accés per a cotxes (de motor).
psv	yes/private/ permissive/ unknown/no		Permís d'accés per al transport publico.
motorboat	yes/private/ permissive/ unknown/no		Permís d'accés per a embarcacions de motor.
boat	yes/private/ permissive/ unknown/no		Permís d'accés per a embarcacions.
oneway	yes (o true o 1)/ no (o false)/ -1		Direcció única permesa en un tram de via. 1 mateixa direcció del segment; -1 direcció

			contraria
noexit	yes		Via sense sortida, només té un accés.
date_on	Data		Data inicial de la restricció
date_off	Data		Data final de la restricció
day_on	Dia de la setmana		Dia de la setmana en que s'inicia la restricció
day_off	Dia de la setmana		Dia de la setmana en que s'acaba la restricció
hour_on	Hora		Hora d'inici de la restricció
hour_off	Hora		Hora de final de la restricció
maxweight	Num		Límit de pes en tones
maxheight	Num		Límit d'altura en metres
maxwidth	Num		Límit d'amplada en metres
maxlength	Num		Límit de longitud en metres
maxspeed	Num		Velocitat màxima en km/h
minspeed	Num		Velocitat mínima en km/h
toll	yes		Peatge

Nomenclatura

o Nom

Clau	Valor	Element	Comentari
name	Definit per l'usuari		Nom comuna por defecte
int_name	Definit per l'usuari		Nom internacional
nat_name	Definit per l'usuari		Nom nacional
reg_name	Definit per l'usuari		Nom regional
loc_name	Definit per l'usuari		Nom local
old_name	Definit per l'usuari		Nom històric
name:lg	Definit per l'usuari		Nom en un altre idioma

o Referències

Clau	Valor	Element	Comentari
ref	Definit per l'usuari		Referència comuna por defecte

int_ref	Definit per l'usuari		Referència internacional
nat_ref	Definit per l'usuari		Referència nacional
reg_ref	Definit per l'usuari		Referència regional
loc_ref	Definit per l'usuari		Referència local
old_ref	Definit per l'usuari		Referència històrica
ncn_ref	Definit per l'usuari		Número de referència de ruta
source ref	Definit per l'usuari		Registre d'URI, referència a la font o a un altre enllaç a fonts físiques.
icao	Definit per l'usuari		Codi internacional d'aeroport OACI
iata	Definit per l'usuari		Codi internacional d'aeroport IATA

o Llocs

Clau	Valor	Element	Comentari
place	continent		Continent.
place	country		País.
place	state		Estat o província.
place	region		Regió.
place	county		Ajuntament.
place	city		Més de 100,000 habitants.
place	town		Entre 10,000 i 100,000 habitants.
place	village		Menys de 10,000 habitants.
place	hamlet		Només algunes cases.
place	suburb		Barri separat d'un poble o ciutat, si té ajuntament propi s'hauria de marcar com a tal.
place	locality		Paratge. Llocs no habitats amb nom propi.
place	island		Illa.
place_name	Definit per l'usuari		Nombre del lloc
place_number	Definit per l'usuari		Número de finca o de portal
Postal_code	Definit per l'usuari		Codi postal
Is_in	Definit per l'usuari		Per a categoritzar.

Anotacions (Annotation)

Estan permeses algunes claus amb algun dels valors següents

Clau	Valor	Element	Comentari
note	Definit per l'usuari		Anotació. Mapa que permet veure anotacions
image	URI		Referència a una imatge
source	extrapolation		Extensió de la informació (extrapolació) a partir d'una font coneguda
source	knowledge		Coneixement local o comú
source	historical		Provinent de mapes sense copyright o altres documents històrics
source	image		Fotografia o vídeo
source	survey		Registre gpx o una altre font geodèsica física
source	voice		Gravacions de veu, p.e. dictàfon
source	Definit per l'usuari		Definit per l'usuari
source_ref	Definit per l'usuari		Registre d'URI, referència a la font o un altre enllaç a fonts físiques.
created_by	Definit per l'usuari		Definit per l'editor utilitzat (JOSM, applet, osmeditor, etc), normalment per tractar els errors Utf8.