Projecte final de carrera:

PROBLEMES DE PROXIMITAT EN XARXES DE CARRETERES

Anna Casas Bosch
Joan Rosset Bordas

Setembre del 2008

Índex

Índex de Figures

1. Introducció

1.1. Motivació del problema

Actualment, és de gran interès en l'àrea de Sistemes d'Informació Geogràfica disposar de software per analitzar problemes de proximitat en xarxes de carreteres.

La distància entre dos punts de la xarxa de carreteres ve definida pel cost del camí mínim entre ells, el qual depèn de la connectivitat i dels pesos de la nostra xarxa. Calcular camins mínims i distàncies en la xarxa és un problema fonamental d'optimització amb importants aplicacions en diferents dominis com Sistemes d'Informació Geogràfica, Serveis de Localització, Sistemes de Navegació, etc.

1.2. Presentació del problema

Per a poder resoldre problemes de proximitat cal modelar una xarxa de carreteres N, que no és res més que un graf no dirigit amb pesos N(V, A, P). El conjunt V, n=|V|, és una col·lecció de vèrtexs que representen interseccions de carreteres (grau superior a 2), punts terminals (grau 1) i punts de forma (grau 2). Els punts de forma són aquelles interseccions que no són punts terminals i que s'utilitzen per a modelar la carretera. El conjunt de vèrtexs V fa referència a un sistema de coordenades determinat. El conjunt A, m=|A|, és una col·lecció d'arestes que representen segments de la carretera, cadascun d'ells connectat a dos vèrtexs. Tota aresta està associada a un pes positiu p(a), que representa, per exemple, la distància euclidiana, el temps, el cost, etc. necessari per a desplaçar-se entre els dos vèrtexs de l'aresta.

Un exemple de xarxa de carreteres el podem veure a la *figura 1.1*, on les interseccions i els punts terminals estan representats amb punts grans i els punts de forma estan representats per punts petits.

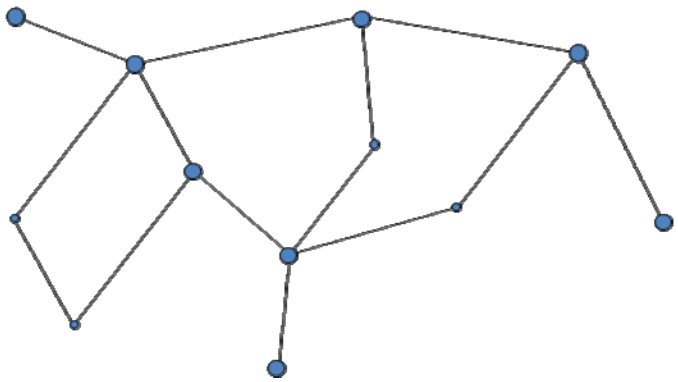


Figura 1.1: Modelat de xarxa de carreteres

En general els grafs que modelen xarxes de carreteres tenen un grau constant i, per tant, no són densos (m=O(n)). En el pitjor cas el graf podria arribar a ser dens ($m=O(n^2)$). Cal destacar que per simplicitat nosaltres modelem xarxes de carreteres com a grafs no dirigits, però els nostres mètodes poden ser fàcilment aplicats a xarxes de carreteres modelades com a grafs dirigits, on cada aresta té un pes diferent per a cada direcció permesa.

En una xarxa de carreteres podem tenir objectes estàtics o facilitats (hotels, gasolineres, restaurants, etc.) i objectes mòbils (cotxes, vianants, etc.).

La distància entre dos punts de la xarxa de carreteres ve definida pel cost del camí mínim entre ells, el qual depèn de la connectivitat i dels pesos de la nostra xarxa. Calcular la distància entre diferents punts tan sols és el primer pas a fer per a poder resoldre la majoria de problemes de proximitat. A partir del càlcul de distàncies es podran calcular, per exemple, els diferents diagrames de Voronoi sobre els objectes de la xarxa. Aquests diagrames permeten donar resposta a problemes de proximitat com trobar els k veïns més propers.

A partir d'una sèrie de facilitats i un conjunt predeterminat de punts de la xarxa, que anomenarem punts de consulta podem resoldre el problema dels k agregats veïns més propers determinant quines k facilitats minimitzen una certa funció de distància respecte als punts de consulta. Per a poder resoldre el problema dels k veïns més propers inversos es partirà d'una sèrie de facilitats i caldrà determinar quines són les k facilitats que es troben més a prop d'una determinada facilitat que de les altres.

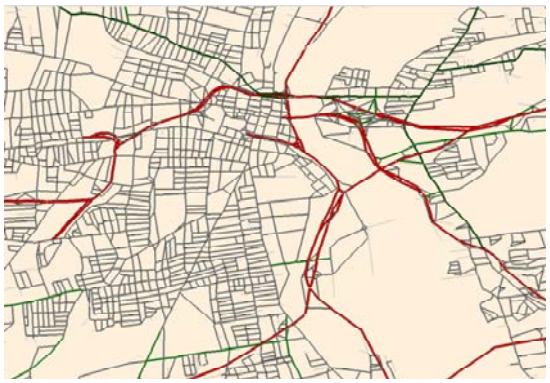


Figura 1.2: Xarxa de carreteres

En aquest entorn es presenten molts problemes, com per exemple:

- Trobar quines són les 5 gasolineres més pròximes a un cotxe.
- Si una pizzeria vol enunciar una nova campanya als seus clients, enviarà el fullets només a aquells clients que tenen el seu restaurant com a la seva pizzeria més propera.
- Si tenim 4 amics que volen anar a un restaurant, trobar aquell que el desplaçament dels amics sigui mínim.

1.3. Objectius del projecte

Degut a la complexitat i a l'abast del projecte, sumats al gran nombre de funcionalitats que cal dissenyar i implementar, es va decidir dividir el projecte en dues parts i desenvolupar-lo conjuntament l'Anna Casas i en Joan Rosset.

Una part dels objectius són comuns i han permès disposar d'unes funcionalitats bàsiques. Una altra part dels objectius són individuals i estan més encarats a la resolució de problemes concrets. Tot i això, per a poder desenvolupar els objectius individuals de cadascú sovint es necessiten altres funcionalitats dissenyades per l'altre membre del grup.

Objectius comuns:

- Recerca de formats de fitxers de xarxes de carreteres.
- Cerca de fitxers del tipus escollit.
- Llegir la informació d'una xarxa de carreteres des de fitxer.
- Visualitzar xarxa de carreteres.
- Posicionar objectes estàtics i mòbils.
- Visualitzar la informació d'aquests objectes.
- Eliminar objectes posicionats a la xarxa.
- Guardar en un fitxer tots els objectes posicionats.
- Crear una interfície gràfica i visualitzar els resultats de totes les funcionalitats.

Objectius individuals Anna:

- Generar parametrització de la xarxa de carreteres.
- Determinar camins mínims sobre diferents punts de la xarxa.
- Calcular funció camp distàncies.
- Crear una estructura geomètrica que representi informació de proximitat sobre un conjunt de punts o objectes (diagrama de Voronoi proper, llunyà i ordre k).
- Resoldre problema de proximitat k veïns propers.
- Resoldre problema de proximitat agregat k veïns propers.
- Calcular punts d'interès de la xarxa.

Objectius individuals Joan:

- Crear estructura de dades per distribuir els vèrtexs sobre el pla 2D (QuadTree comprimit de vèrtexs).
- Crear estructura de dades per distribuir els objectes estàtics en el pla 2D (QuadTree comprimit d'objectes).
- Crear estructura de dades per distribuir les arestes en el pla 2D (RTree d'arestes).
- Seleccionar subxarxa de carreteres.
- Calcular cercle amb centre a un punt de la xarxa i un radi donat.
- Resoldre problema de proximitat inversos k veïns propers.

NOTA IMPORTANT

Inicialment es volien crear dues documentacions separades, una per a cada membre del grup, on figuressin els objectius individuals. Durant el desenvolupament de la memòria s'ha vist que molts dels conceptes i de les funcionalitats dels dos membres del grup eren dificilment separables. És per aquest motiu, degut a aquesta estreta vinculació, que finalment s'ha pres la decisió de realitzar una única memòria conjunta.

El motiu principal que ens ha portat a unir les dues memòries ha estat facilitar la comprensió del treball fet als lectors d'aquesta memòria. Tal i com s'ha explicat en aquest apartat, es poden diferenciar clarament els objectius comuns i els objectius individuals de cadascú.

1.4. Abast del projecte

Després de fer un estudi previ i definir els objectius, s'han recollit els requeriments del projecte i s'ha fet l'anàlisi i el disseny utilitzant una metodologia orientada a objectes. Per desenvolupar l'aplicació s'ha treballat bàsicament amb C++, Qt4, OpenGL i també s'ha fet servir la potència de la GPU per a accelerar els càlculs.

L'objectiu final del projecte és poder resoldre diversos problemes de proximitat. És important entendre i saber resoldre aquests problemes, però també és molt important poder-los resoldre de forma ràpida i eficient.

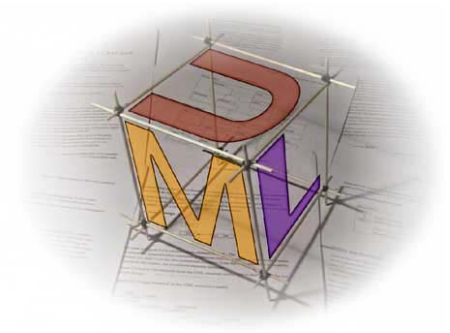
1.5. Metodologia UML

En el desenvolupament del projecte s'ha utilitzat la metodologia UML (Unified Modeling Language). És un dels llenguatges de modelat de sistemes de software més conegut i utilitzat en l'actualitat.

És un llenguatge que permet modelar, construir i documentar els elements que formen un sistema de software orientat a objectes. S'ha convertit pràcticament en un estàndard degut a que ha estat impulsat pels autors dels mètodes més utilitzats d'orientació a objectes. En la seva creació també han participat empreses de gran pes com Microsoft, Hewlett-Packard, Oracle o IBM.

Aquesta notació ha estat àmpliament acceptada degut al prestigi dels seus creadors i degut a la incorporació dels principals avantatges de cada un dels mètodes particulars en els que es basa. UML ha posat fi a la guerra dels mètodes que va haver-hi al llarg dels anys 90. Amb la metodologia UML s'ha fusionat la notació de diferents

tècniques per a formar una eina compartida entre tots els enginyers del software que treballen en el desenvolupament orientat a objectes.



1.6. Organització d'aquest document

Aquesta memòria s'ha organitzat en 12 capítols:

- **Capítol 1, Introducció:** En aquest capítol es pretén explicar la problemàtica que ha portat a la realització del projecte i, a partir d'aquesta, explicar quins són els seus objectius i abast. També es dóna una visió de la metodologia utilitzada i de la organització d'aquest document.

- **Capítol 2, Estudi previ:** En aquest capítol s'explicaran les principals tasques desenvolupades durant les primeres etapes de realització del projecte. Ha estat necessari l'estudi previ d'alguns conceptes així com l'aprenentatge d'alguns llenguatges de programació per a poder desenvolupar l'aplicació.

- **Capítol 3, Requeriments:** En aquest capítol seran definits els *Requeriments del programari*, els quals recullen, a grans trets, els objectius de l'aplicació juntament amb les seves funcionalitats desitjades. Aquest document ens ha de permetre entendre els elements que envoltaran el sistema informàtic que s'intenta construir, com són: les persones, els procediments, el hardware i el propi software.

- **Capítol 4, Anàlisi:** L'objectiu de l'anàlisi és obtenir una comprensió precisa de les necessitats del sistema, és a dir, s'encarrega de la investigació del problema a resoldre (*que*), però no s'interessa en trobar-ne una solució (*com*). En aquest procés ens ocupem de traduir els requeriments comentats en el capítol anterior a un llenguatge més formal segons l'Enginyeria del Programari.

- **Capítol 5, Disseny:** En aquesta etapa de *disseny del sistema* s'intenta adaptar la documentació generada en el capítol anterior en vistes a la implementació final del sistema mitjançant diverses eines de programació orientada a objectes.

- **Capítol 6, Implementació:** Dins d'aquest capítol es pretén donar a conèixer els mètodes implementats en l'aplicació final que es consideren més importants pel funcionament d'aquesta seguint un cert ordre coherent.

- **Capítol 7, Resultats:** En aquest capítol es mostra alguna prova d'execució de l'aplicació mostrant els resultats amb diferents mides de mapes.

- **Capítol 8, Software:** En aquest capítol s'esmenten les llibreries i el software utilitzats per tal de dur a terme aquest projecte. Es mostren les eines que han permès la implementació de l'aplicació així com els programes necessaris per construir la documentació corresponent.

- **Capítol 9, Conclusions:** En aquest capítol s'exposaran les conclusions i la planificació del projecte. També s'hi expliquen els possibles treballs futurs.

- **Capítol 10, Manual d'usuari:** En aquest capítol es mostra com realitzar cadascuna de les funcionalitats de l'aplicació d'una manera senzilla i il·lustrativa.

- **Capítol 11, Agraïments:** En aquest capítol s'esmenten els agraïments personals de l'autor.

- **Capítol 12, Bibliografia:** En aquest capítol, ja concloent la documentació, s'exposen totes les referències utilitzades per a la realització del projecte.

2. Estudi Previ

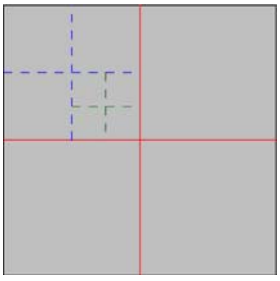
En aquest capítol s'explicaran les principals tasques desenvolupades durant les primeres etapes de realització del projecte. Ha estat necessari l'estudi previ d'alguns conceptes així com l'aprenentatge d'alguns llenguatges de programació per a poder desenvolupar l'aplicació.

2.1. Conceptes teòrics

Cal tenir clars alguns conceptes teòrics per a poder entendre algunes de les funcionalitats de l'aplicació i el seu funcionament. A continuació, s'explicaran els conceptes teòrics més importants. Més endavant, en el capítol d'implementació s'hi farà referència i s'exposaran detalls de com s'ha fet la seva implementació ja que molts d'aquests caldrà implementar-los utilitzant la potència de la GPU per agilitzar els càlculs.

2.1.1. QuadTree

Un QuadTree és una estructura de dades molt utilitzada en informàtica gràfica i es basa en una descomposició jeràrquica dels objectes de l'espai que descriuen. Com s'ha explicat al capítol anterior, els objectes que es necessiten per a crear una xarxa de carreteres estan ubicats mitjançant un sistema de coordenades. Per tant, els QuadTrees que es creen a l'aplicació serviran per a ordenar aquestes coordenades a l'espai 2D.



Un QuadTree es crea a partir d'una caixa englobant de forma quadrada que conté tots els punts (coordenades) que s'estan descrivint. Es divideix aquest espai que hi ha en quatre quadrats iguals (quadrants) generant quatre fills que, juntament amb el pare, formen un arbre 4-ari. Tot seguit es reparteixen tots els punts en el fill corresponent depenent de la seva posició. Aquest procés es va repetint fins que cada node concret conté tan sols un punt.

L'ordre que s'estableix per a assignar quin número té cada fill s'estableix segons l'ordre de Morton, que dóna prioritat al fill superior-esquerre.

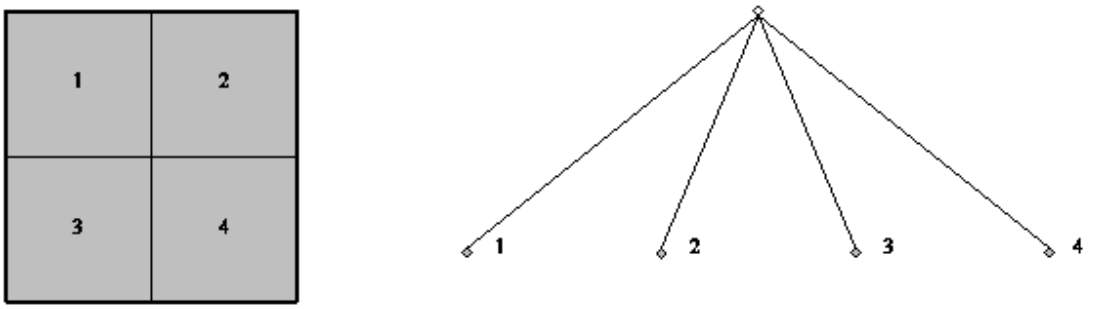


Figura 2.1: QuadTree: Ordre de Morton

A la *figura 2.2* podem veure un exemple molt senzill d'un QuadTree amb 5 punts juntament amb el seu arbre generat.

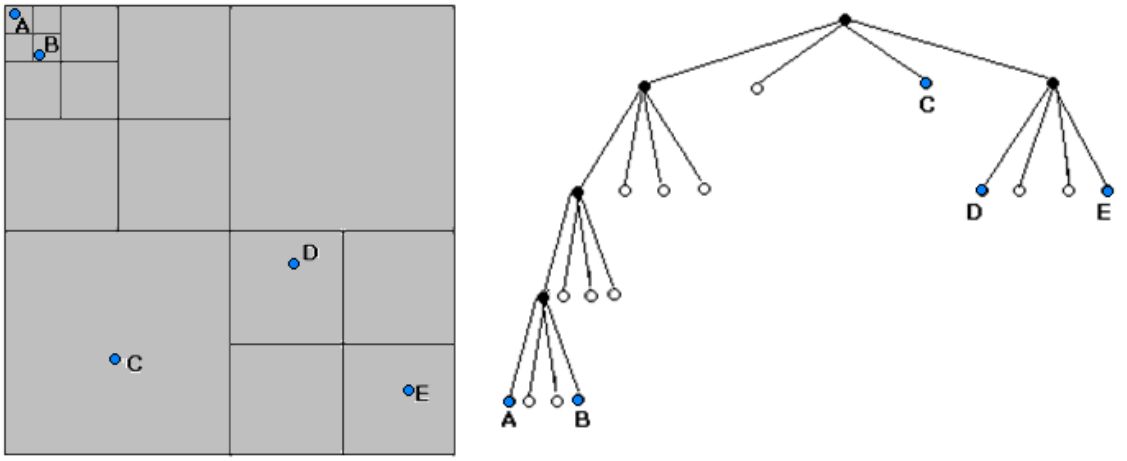


Figura 2.2: QuadTree

2.1.2. QuadTree comprimit

Un QuadTree comprimit és una millora que es pot aplicar al QuadTree normal, que incrementa la seva complexitat però el fa molt més eficient.

Aquest QuadTree només agafa els quadrants més interessants de l'arbre ignorant els demés. S'entén com a quadrant interessant aquell que té més d'un fill no buit.

A la següent figura es pot visualitzar el QuadTree comprimit corresponent a la compressió del QuadTree de l'apartat anterior.

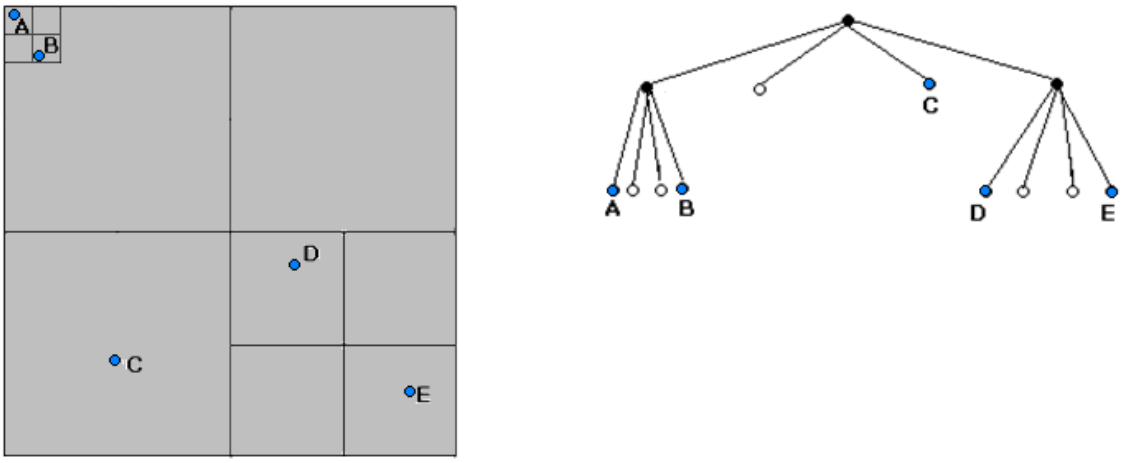


Figura 2.3: QuadTree comprimit

Observant la *figura 2.2* del QuadTree normal es pot comprovar que, depenent de la posició dels punts, es poden generar branques desagradables que ens provoquen lentitud alhora de realitzar cerques. En canvi, utilitzant el nou QuadTree comprimit s'eviten aquestes branques deixant només aquelles que de debò interessen.

2.1.3. RTree

Un RTree és una estructura de dades en forma d'arbre que s'utilitza per a indexar informació multidimensional d'objectes arbitraris. És un arbre balancejat amb registres indexats a les fulles que contenen punters als objectes. A l'aplicació aquests objectes seran les arestes dels grafs al pla 2D.

Un RTree es crea a partir d'una caixa englobant de forma rectangular que conté tots els objectes (arestes) que s'estan descrivint. Tot seguit, aquesta estructura de dades divideix l'espai de forma jeràrquica en conjunts, que poden ser sobreposats.

A la següent figura es mostra un exemple d'un RTree per tal de comprendre millor el seu funcionament.

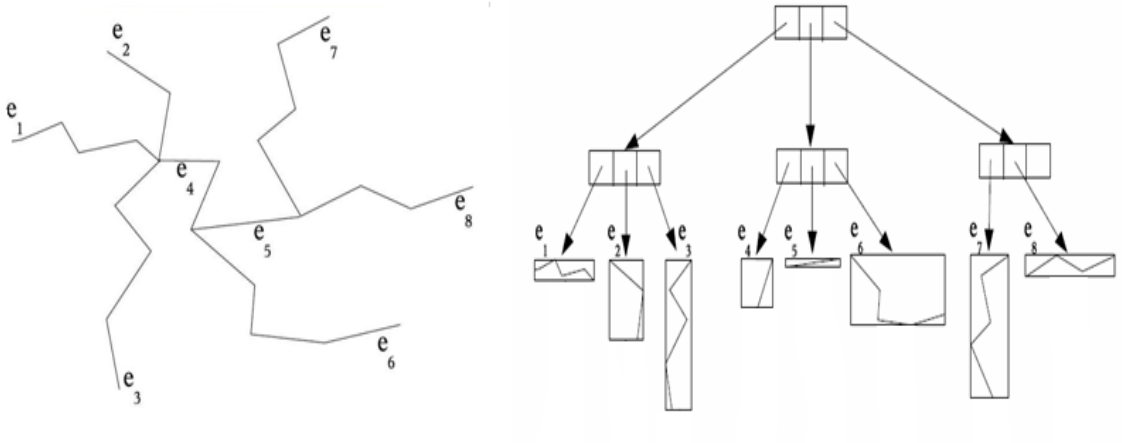


Figura 2.4: RTree

La tècnica que utilitza l'RTree per a dividir l'espai en forma jeràrquica és la següent:

- La caixa englobant (rectangle) de cada aresta de la xarxa de carreteres és l'objecte espaial que contindrà l'RTree.
- Els objectes que es troben més a prop s'ajunten formant un nou node en forma rectangular.
- Els nodes es van agrupant recursivament seguint el mateix principi fins arribar a l'arrel de l'arbre.

Així doncs, la part més important de la construcció d'un RTree és el moment que s'han d'anar agrupant els objectes i els nodes. Per a fer una bona agrupació cal que la àrea del rectangle resultant menys les àrees dels objectes que conté sigui el més petita possible.

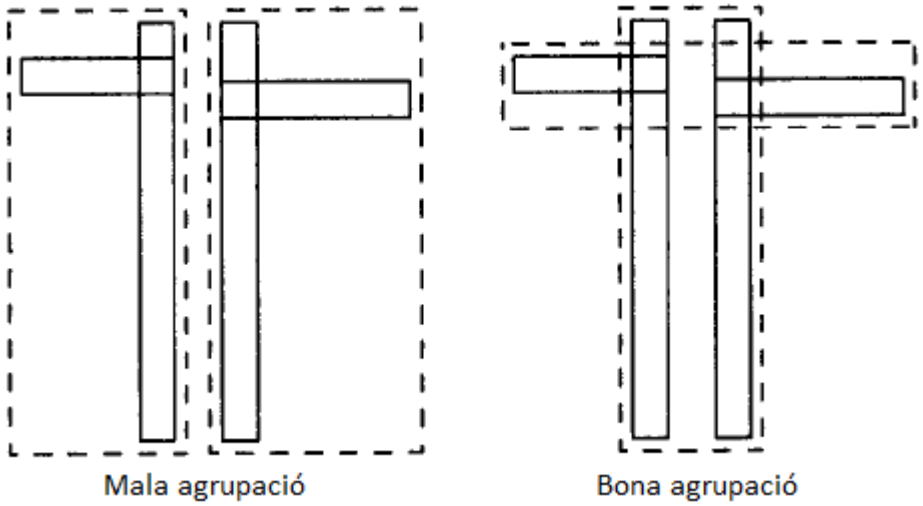


Figura 2.5: RTree: Agrupació

Com es pot observar a la *figura 2.5*, l'àrea de les agrupacions de la segona imatge és menor a la agrupació de la primera. A més, es pot comprovar que a la segona imatge la unió de les zones queden sobreposades. Aquest fet no té cap repercussió negativa a l'RTree.

2.1.4. Parametrització de la xarxa de carreteres

Quan parlem de calcular la parametrització de la xarxa de carreteres ens referim a compactar la xarxa per tal de poder fer els càlculs d'una manera més eficaç i precisa.

Per calcular la parametrització necessitem tenir totes les arestes del graf. Crearem una reixa rectangular de punts on hi mapejarem cada una d'aquestes arestes ocupant, cada una, un nombre determinat de punts. La reixa tindrà una mida determinada, per tant, caldrà calcular el nombre de punts que s'assignaran a cada aresta segons la seva longitud real. S'aniran mapejant les arestes a la reixa, una al costat de l'altra, de manera que mai quedin dues arestes superposades ni que cap aresta quedi tallada.

A la *figura 2.6* podem fer-nos una idea gràfica de com funciona la parametrització de la xarxa.

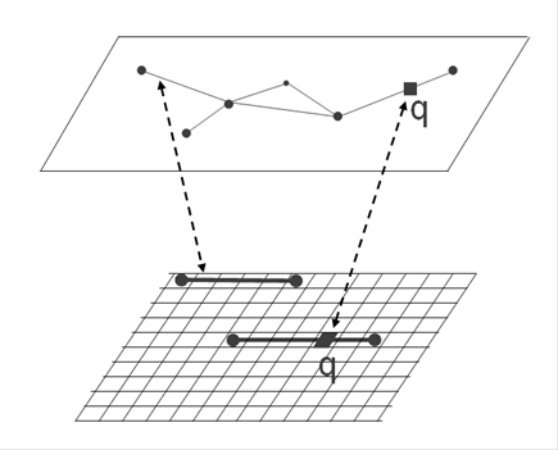


Figura 2.6: Parametrització de la xarxa

2.1.5. Càlcul del camí mínim: Dijkstra

El càlcul de la distància mínima entre dos punts de la xarxa de carreteres es realitza mitjançant l'algorisme de camins mínims de Dijkstra.

Veiem com funciona aquest algorisme amb un petit exemple. Calcularem el camí mínim des d'un punt de la xarxa fins a tots els altres. Per a fer-ho cada vèrtex haurà de guardar el cost del camí mínim per arribar fins a ell i el seu vèrtex antecessor per a posteriorment poder recórrer el camí mínim trobat. També necessitarem un vector on guardarem tots aquell vèrtexs als quals es pot arribar i que encara no tenen cost mínim assignat vector={}.

1. Tenim un graf com el que es mostra a la *figura 2.7*. Donem nom a cada un dels vèrtexs i assignem el pes a cada aresta. Per començar l'algorisme cal escollir un vèrtex inicial, en el nostre cas, escollim el vèrtex. Vector={A}

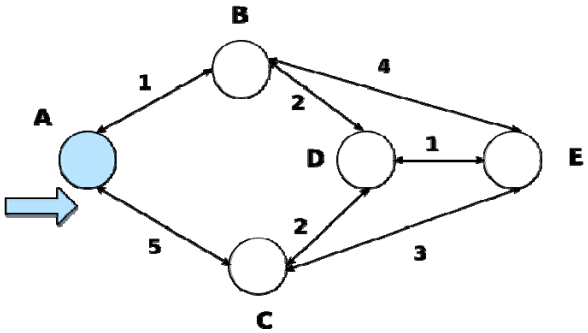


Figura 2.7: Dijkstra: pas 1

2. Inicialment assignarem cost infinit i antecessor nul a cada un dels vèrtexs del graf, menys al vèrtex A que és l'inicial i té cost 0. Des del vèrtex A podem arribar al vèrtex B amb cost 1 i al vèrtex C amb cost 5, així es pot veure indicat a la figura 2.8. Hem de treure el vèrtex A del vector i afegir-hi els dos vèrtexs nous als quals podem accedir vector={B,C}. Ara, entre els vèrtexs que tenim al vector cal que n'escolli'm un, el que té menor cost, per tant, el vèrtex B.

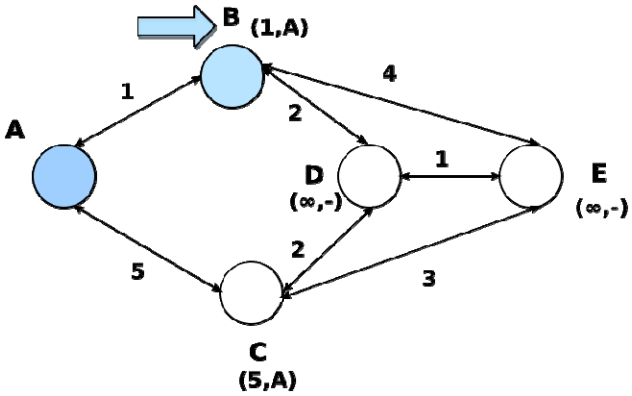


Figura 2.8: Dijkstra: pas 2

3. Des del vèrtex B podem arribar al vèrtex E amb cost 5 i al vèrtex D amb cost 3. Hem de treure el vèrtex B del vector i afegir-hi els nous vèrtexs als quals podem accedir vector={C,D,E}. Entre els vèrtexs que tenim ara al vector, el que té menor cost és el D, per tant, el seleccionem.

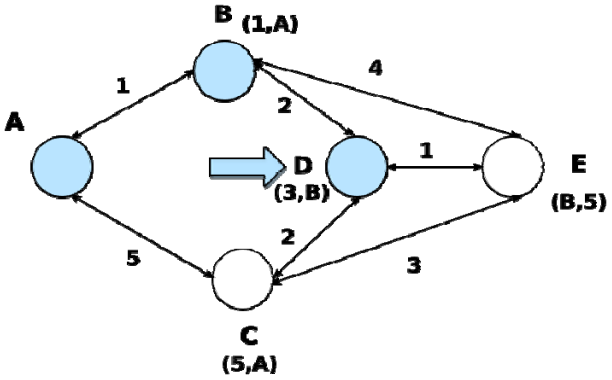


Figura 2.9: Dijkstra: pas 3

4. Des del vèrtex D podem arribar al vèrtex E amb cost 4 i al vèrtex C amb cost 5. Al vèrtex E ja s'hi podia arribar abans però com que el cost amb el qual s'hi arriba des de D és menor l'actualitzem. Al vèrtex C també s'hi podia arribar abans però com que el cost amb el qual s'hi arriba des de D és el mateix que tenia no el canviem. Hem de treure el vèrtex D del vector i no cal afegir-n'hi cap vector={C,E}. Escollim el vèrtex amb menor cost, el vèrtex E.

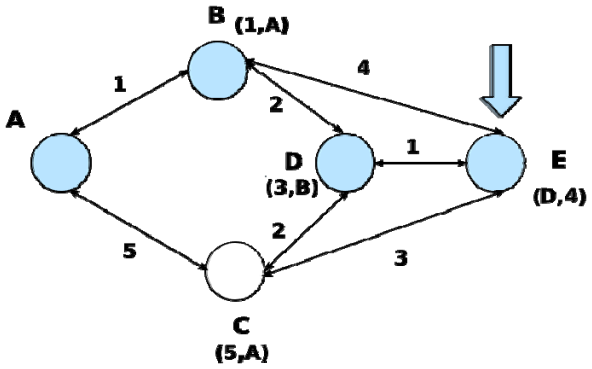


Figura 2.10: Dijkstra: pas 4

5. Des del vèrtex E podem accedir al vèrtex C però com que ja s'hi accedia abans amb menor cost no el modifiquem. Hem de treure el vèrtex E del vector i no cal afegir-n'hi cap vector={C}. Escollim l'únic vèrtex que queda, el C.

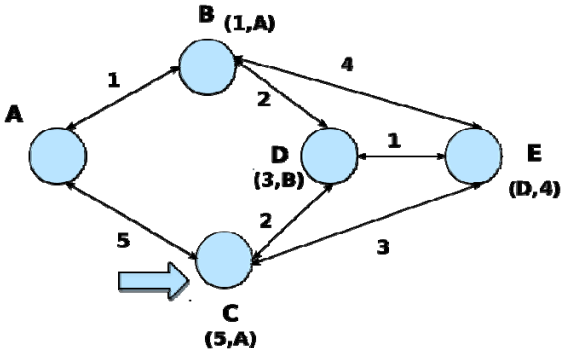


Figura 2.11: Dijkstra: pas 5

6. Des del vèrtex C ja no es pot accedir a cap vèrtex nou. Hem de treure el vèrtex C del vector i com que aquest queda buit vol dir que ja hem acabat l'algorisme, i per tant, hem trobat el camí mínim des del vèrtex inicial fins a tots els vèrtexs als quals es pot accedir.

2.1.6. Funció camp de distàncies

La funció camp de distàncies determina quina és la distància des d'una determinat objecte fins a qualsevol punt de la xarxa.

La distància entre dos punts de la xarxa ve determinada pel cost del camí mínim entre ells. El càlcul de la distància mínima entre dos punts de la xarxa de carreteres es realitza mitjançant l'algorisme de camins mínims de Dijkstra.

Així, per obtenir la funció distància d'un objecte caldrà aplicar l'algorisme de Dijkstra i trobar el camí mínim per arribar a cada un dels vèrtexs de la xarxa. Després, serà necessari discretitzar la funció distància per obtenir la distància per a cada un dels punt interiors de les arestes del graf.

Podem veure la funció distància a la *figura 2.12*.

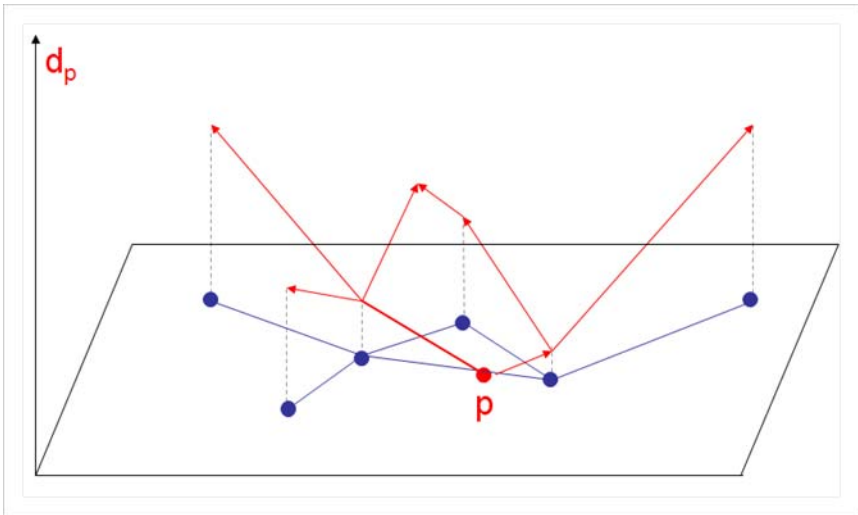


Figura 2.12: Funció distància

2.1.7. Diagrames de Voronoi

Quan es parla dels diagrames de Voronoi es fa referència a una estructura geomètrica que representa informació de proximitat sobre un conjunt de punts. Els diagrames de Voronoi són una de les estructures fonamentals dins de la Geometria Computacional, d'alguna manera emmagatzemen tota la informació referent a la proximitat entre punts.

Però què és un diagrama de Voronoi? Per tal d'entendre més ràpidament el concepte introduirem els diagrames de Voronoi sobre el pla.

En la següent imatge podem veure que el punt x està dins de la regió de punts més propers a p. Aquesta regió és la regió de Voronoi del punt p, i p és el punt generador de la regió.

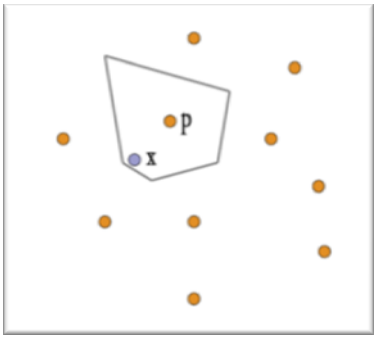


Figura 2.13: Regió Voronoi

La unió de totes les regions de Voronoi formen el diagrama de Voronoi, tal i com podem veure a la *figura 2.14*.

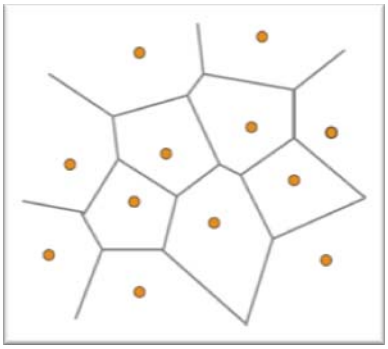


Figura 2.14: Diagrama de Voronoi

2.1.7.1. Propietats del diagrama de Voronoi

Podem destacar les següents propietats d'un diagrama de Voronoi:

- Dos punts p_i i p_j són veïns si comparteixen una aresta de la regió.
- Un vèrtex és un punt equidistant a tres punts generadors i és la intersecció de tres arestes (si el punt és equidistant a més de tres generadors parlarem de casos degenerats).
- Els punts d'una aresta de Voronoi són equidistants als dos punts generadors de les dues regions de les quals formen part.
- Dins del cercle amb centre un vèrtex de Voronoi i que passa per tres punts generadors no pot existir cap altre punt generador.

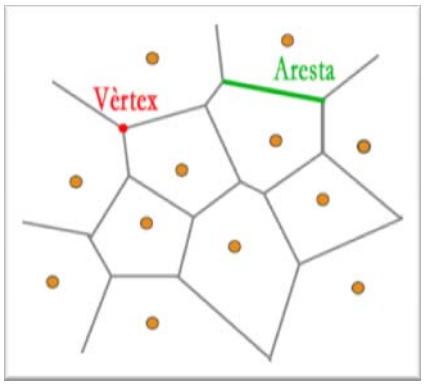


Figura 2.15: Diagrama de Voronoi: vèrtex i aresta

2.1.7.2. Aplicacions del diagrama de Voronoi

Veiem algunes de les aplicacions que pot tenir aquest diagrama:

- El veïns més propers.

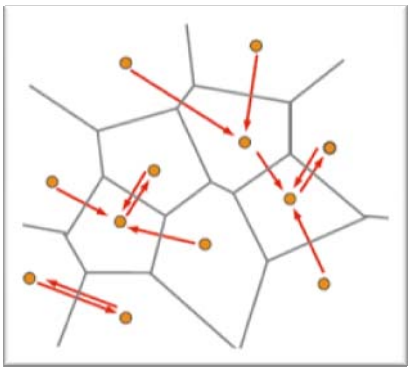


Figura 2.16: Diagrama de Voronoi: veïns propers

- Major cercle possible. Es tracta de trobar el major cercle buit possible amb centre dins de l'envolvent convexa. El centre del cercle ha de ser un vèrtex del diagrama de Voronoi o els punts d'intersecció entre l'envolvent convexa i els eixos de Voronoi.

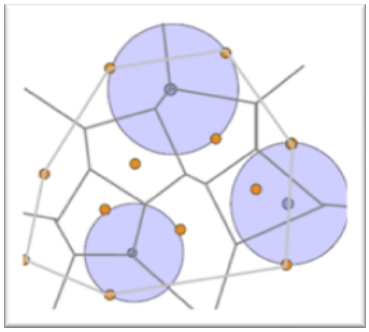


Figura 2.17: Diagrama de Voronoi: major cercle possible

- Triangulació. El diagrama de Voronoi proporciona la millor triangulació possible (Triangulació de Delaunay).

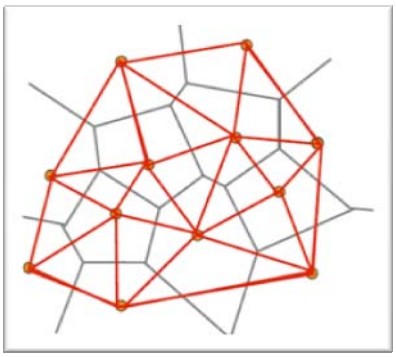


Figura 2.18: Diagrama de Voronoi: triangulació

El diagrama de Voronoi permet donar resposta a molts problemes. On i quantes antenes de telefonia mòbil cal col·locar perquè la recepció de la senyal sigui completa en una regió? On col·loquem un nou comerç per tal que sigui competent?

2.1.7.3. Diagrama de Voronoi proper, llunyà i ordre k

Un cop entès el concepte del que és un diagrama de Voronoi sobre el pla, podem aplicar els mateixos conceptes sobre la xarxa de carreteres. Anem a veure els diferents tipus de diagrames de Voronoi que caldrà generar sobre la xarxa de carreteres.

A partir de la funció distància (explicada a l'apartat anterior) de cada punt es poden obtenir diferents diagrames de Voronoi: el proper, el llunya i l'ordre k. En la *figura 2.19* podem veure 4 punts: blau, vermell, verd i groc que considerem punts generadors i dels quals tenim representades la seva funció distància. A més, hem afegit 4 punts qualsevol anomenats a, b, c i d, que ens ajudaran a entendre els diferents tipus de diagrames a través d'exemples.

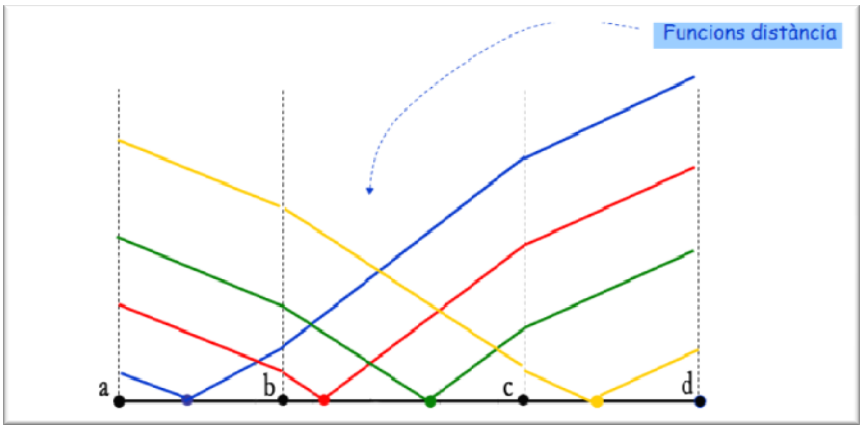


Figura 2.19: Funcions camp distància

2.1.7.3.1. Diagrama de Voronoi proper d'ordre 1

El diagrama de Voronoi proper (o proper d'ordre 1) ens determina per a cada punt qualsevol de la xarxa quin és el seu punt generador més proper. Per tal de trobar aquest diagrama haurem d'utilitzar la primera capa que es forma amb les funcions distància, tal i com es pot veure en la següent figura. Cada zona pintada d'un determinat color és la regió de Voronoi del punt generador que té el mateix color, per tant, tots els punts que queden més propers a aquest punt generador. Com a exemples, el punt a té com a punt generador més proper el blau, el punt b té com a punt generador més proper el vermell i els punts c i d tenen com a punt generador més proper el groc.

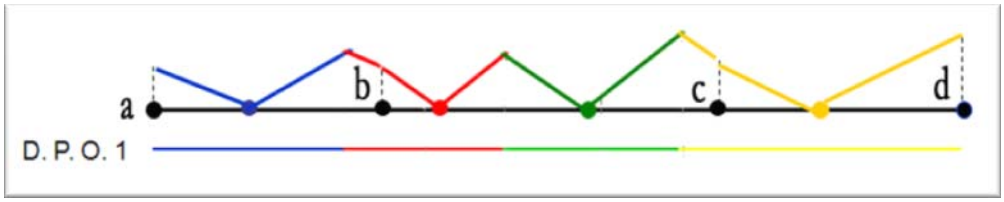


Figura 2.20: Diagrama de Voronoi proper d'ordre 1

2.1.7.3.2. Diagrama de Voronoi proper d'ordre 2

El diagrama de Voronoi proper d'ordre 2 ens determina per a cada punt qualsevol de la xarxa quin és el seu segon punt generador més proper. Per tal de trobar aquest diagrama enlloc d'agafar la primera capa que es forma amb les funcions distància agafem la segona, tal i com es veu en la *figura 2.21*. Cada zona d'un determinat color ens indica que tots els punts d'aquella zona tenen com a segon punt generador més proper el que té el mateix color. Com a exemples, el punt a té com a segon punt generador més proper el vermell, el punt b té com a segon punt generador més proper el blau i els punts c i d tenen com a segon punt generador més proper el verd.

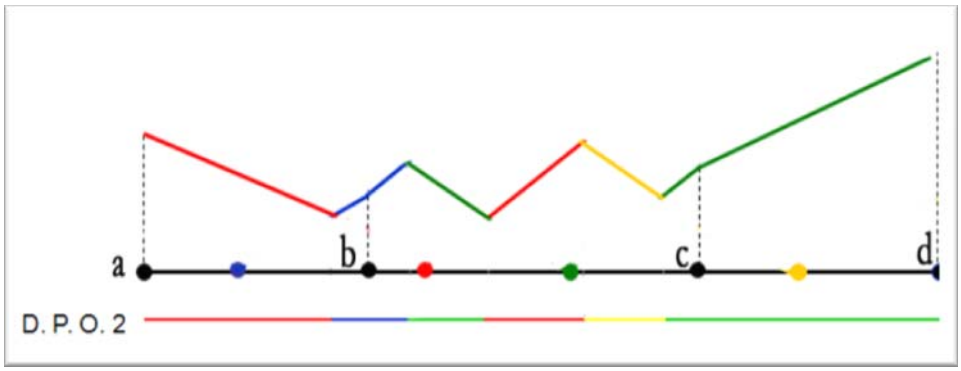


Figura 2.21: Diagrama de Voronoi proper d'ordre 2

Podríem anar agafant les capes successives, tantes com nombre k de punts generadors tenim, i trobar els k diagrames.

2.1.7.3.3. Diagrama de Voronoi llunyà

El diagrama de Voronoi llunyà ens determina per a cada punt qualsevol de la xarxa quin és el seu punt generador més llunyà. Per tal de trobar aquest diagrama agafarem la última capa que es forma amb les funcions distància. Com a exemples, els punts a i b tenen com a punt generador més llunyà el groc i els punts c i d tenen com a punt generador més llunyà el blau.

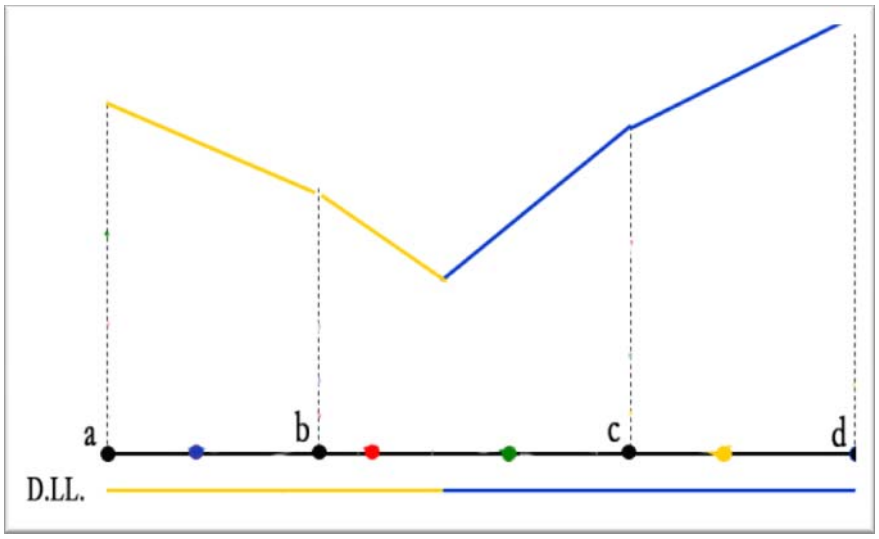


Figura 2.22: Diagrama de Voronoi llunyà

2.1.7.3.4. Diagrama de Voronoi d'ordre k

A partir dels diagrames de Voronoi propers podem obtenir el diagrama de Voronoi d'ordre k. Veurem què és a través d'un exemple. Si volem obtenir el diagrama de Voronoi d'ordre 2, en realitat el que estem buscant són els 2 punts generadors més propers per a cada punt. Per obtenir el diagrama de Voronoi d'ordre 2 necessitem els diagrames de Voronoi proper d'ordre 1 i d'odre 2, tal i com es pot veure en la *figura 2.23*:

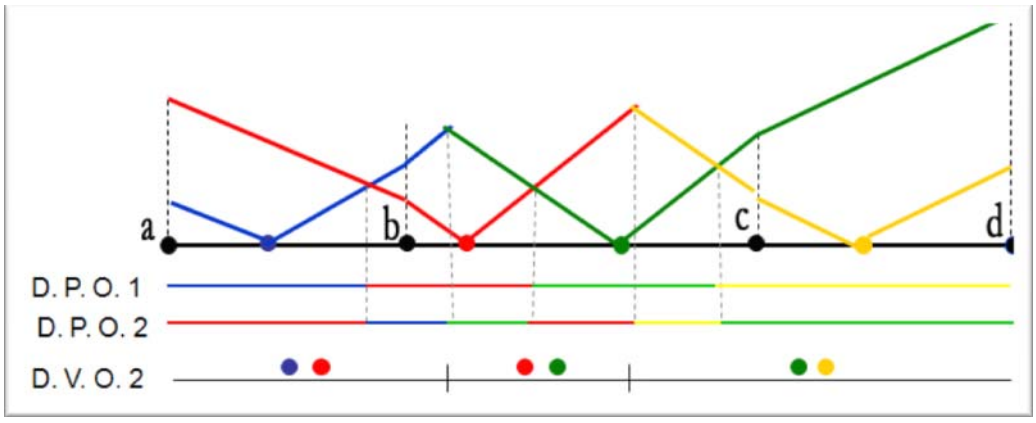


Figura 2.23: Diagrama de Voronoi d'odre k

En el diagrama de Voronoi d'ordre 2 es generen tres zones resultants, cadascuna amb els seus dos colors que indiquen els 2 punts generadors més propers. Com a exemples, els punts a i b tenen com a 2 punts generadors més propers el blau i el vermell i els punts c i d tenen com a 2 punts generadors més propers el verd i el groc.

2.1.8. Calcular el cercle

Un cercle queda determinat per un punt de la xarxa que anomenarem centre i un nombre positiu que anomenarem radi. Quan parlem de calcular un cercle ens referim a trobar tots els punts de la xarxa que disten del centre com a màxim el radi.

En la *figura 2.24* podem veure 2 punts, blau i vermell, dels quals tenim representades la seva funció distància. A més, hem afegit 4 punts qualsevol anomenats a, b, c i d, que ens ajudaran a entendre el problema. Escollim el punt vermell com a centre i introduïm un radi r per tal de generar el cercle. Com es pot veure en la imatge tots els punts de la funció distància que queden per sota la línia del radi ens generen el cercle resultant. Veiem que dels punts a,b, c i d tan sols el punt b queda dins el radi del cercle.

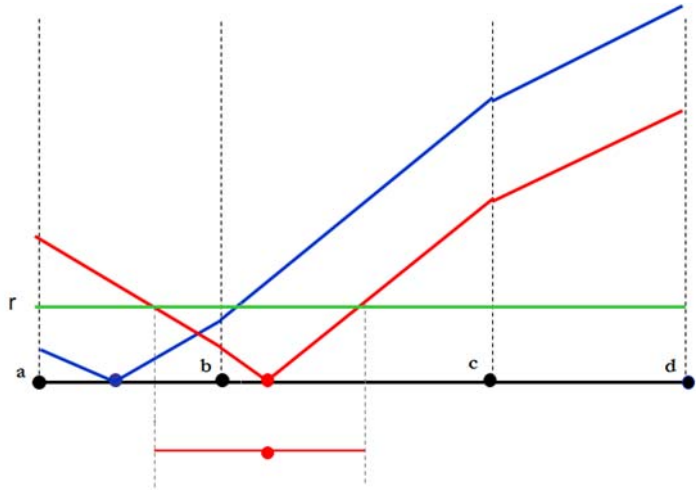


Figura 2.24: Cercle

2.1.9. Punts d'interès

Els punts d'interès representen punts estratègics sobre la xarxa i donen resposta a preguntes com aquesta: Quin és el lloc on hauríem de construir un hospital si volem tenir en compte la localització de les àrees més populars de la ciutat?

Els punts d'interès 1-Center i 1-Median s'utilitzen per a col·locar un nou servei que la gent desitja tenir a prop com pot ser un hotel, un hospital, una gasolinera, un centre de distribució, etc. Si es vol que aquest nou servei estigui el més a prop possible del servei ja existent més allunyat s'utilitzarà l'1-Center i si es vol que el nou servei estigui on la suma de distàncies als ja existents sigui mínima s'utilitzarà l'1-Median.

Els punts d'interès 1-Center Obnoxious i 1-Median Obnoxious serien els oposats als anteriors. S'utilitzen per a col·locar un nou servei que la gent desitja tenir el més lluny possible com pot ser una central nuclear, magatzems de residus perillosos, magatzems de productes pirotècnics, etc.

Es poden calcular els punts d'interès a partir dels camps de distàncies i del diagrama de Voronoi. Depenent del punt que estem buscant el càlcul serà diferent:

- **1-Center**. Ens retorna el punt de la xarxa que minimitza la distància màxima a les facilitats.

  A partir del diagrama de Voronoi llunyà obtindrem el punt amb menor cost que serà l'1-Center.

- **1-Median**. Ens retorna el punt de la xarxa que minimitza la suma de distàncies a les facilitats.

  A partir del suma de tots els camps de distàncies de les facilitats obtindrem el punt amb menor cost que serà l'1-Median.

- **1-Center Obnoxious**. Ens retorna el punt de la xarxa que maximitza la distància mínima a les facilitats.

  A partir del diagrama de Voronoi proper obtindrem el punt amb major cost que serà l'1-Center.

- **1-Median Obnoxious**. Ens retorna el punt de la xarxa que maximitza la suma de distàncies a les facilitats.

  A partir del suma de tots els camps de distàncies de les facilitats obtindrem el punt amb major cost que serà l'1-Median.

2.1.10. Problema de proximitat k-NN

La resolució del problema de proximitat k-NN (k-Nearest Neighbor) o k veïns propers ens permet trobar els objectes més propers a un determinat punt de la xarxa i donar resposta a preguntes com aquestes: Quins són els 5 hotels més a prop de la meva posició? Quin és l'hospital més proper a casa meva?

Es pot resoldre el problema k-NN a partir del diagrama de Voronoi d'ordre k explicat en l'apartat anterior.

Imaginem que volem trobar quines són les 3 gasolineres més properes a la meva posició. Caldria calcular el diagrama de Voronoi d'ordre 3 utilitzant com a punts generadors totes les gasolineres de la xarxa. Seguidament, caldria veure quins són els 3 punts generadors resultants per a la meva posició, és a dir, quines són les 3 gasolineres més properes a la meva posició.

A la *figura 2.25* podem en podem veure el resultat. Els punts de color vermell són gasolineres i el punt verd és la meva posició. El diagrama de Voronoi d'ordre 3 ens dona com a gasolineres més properes les 3 que es veuen ressaltades.

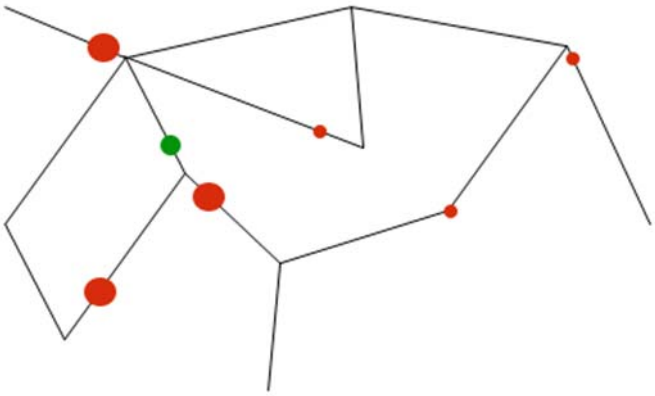


Figura 2.25: Problema k-NN

2.1.11. Problema de proximitat Ak-NN

La resolució del problema de proximitat Ak-NN (Aggregate k-Nearest Neighbor) o k veïns més propers agregats ens permet trobar els k objectes millor situats d'acord amb una funció de distàncies agregades respecte d'una sèrie de punts i donar resposta a preguntar com aquestes: Quin és el restaurant que permet a 5 amics trobar-se recorrent cada un d'ells la mínima distància possible?

Es pot resoldre el problema Ak-NN a partir dels camps de distàncies.

Per tal d'entendre més ràpidament el concepte introduirem els Ak-NN sobre el pla, encara que nosaltres després ho apliquem sobre la xarxa de carreteres.

Imaginem que tenim una sèrie de punts de consulta i una sèrie de facilitats tal i com es mostra en la *figura 2.26* i volem trobar les k facilitats que compleixin una certa funció respecte els punts de consulta. Tenim 3 facilitats representades de color taronja p1, p2, p3 i 2 punts de consulta representats de color verd a i b. Podem veure que les facilitats més properes al punt de consulta a són p3, p2, p1 i que les facilitats més properes al punt b són p1, p2, p3.

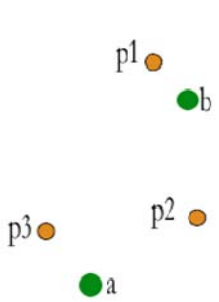


Figura 2.26: Problema Ak-NN

Depenent de la funció a utilitzar el càlcul del problema Ak-NN serà diferent. Imaginem que volem trobar 1 facilitat resultant en cadascun dels problemes:

- **MinMax**. Es tracta de trobar les k facilitats que fan mínima la màxima distància als punts de consulta. S'ha de calcular per a cada facilitat quin és el punt de consulta que es troba a major distància i posteriorment obtenir les k facilitats que tenen el punt de consulta més llunyà a menor distància.

  Fem el càlcul per a l'exemple de la *figura 2.26*. El punt de consulta més llunyà de p1 és a, el punt de consulta més llunyà de p2 és b i el punt de consulta més llunyà de p3 és b. Com que volem obtenir una facilitat, aquella que fa mínima la màxima distància als punts de consulta, agafem la que té una distància menor, que tal i com podem veure a la figura següent és p2.

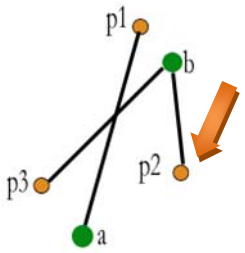


Figura 2.27: Problema Ak-NN: MinMax

- **MinSum**. Es tracta de trobar les k facilitats que fan mínima la suma de distàncies als punts de consulta. S'ha de calcular per a cada facilitat la suma de distàncies als punts de consulta i posteriorment obtenir les k facilitats que fan mínima la suma de distàncies.

  Fem el càlcul per a l'exemple de la *figura 2.26*. Si sumem les distàncies de p1 als dos punts de consulta tenim un valor de 4'3, si sumem les distàncies de p2 als dos punts de consulta tenim un valor de 3'8 i si sumem les distàncies de p3 als dos punts de consulta tenim un valor de 4'1. La facilitat amb la suma mínima (tal i com es pot observar en les següents imatges), i per tant, la facilitat resultant és p2.

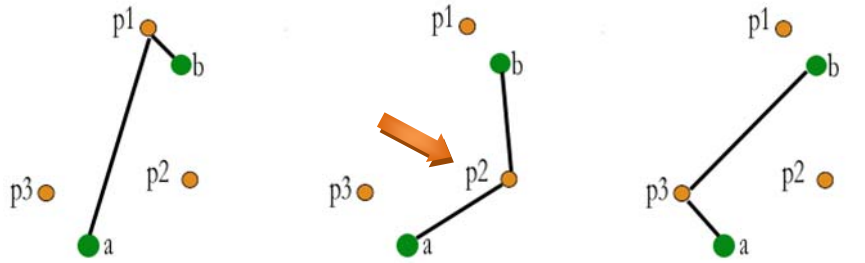


Figura 2.28: Problema Ak-NN: MinSum

- **MaxMin**. Es tracta de trobar les k facilitats que fan màxima la mínima distància als punts de consulta. S'ha de calcular per a cada facilitat quin és el punt de consulta que es troba a menor distància i posteriorment obtenir les k facilitats que tenen el punt de consulta més proper a major distància.

Fem el càlcul per a l'exemple de la *figura 2.26*. El punt de consulta més proper de p1 és b, el punt de consulta més proper de p2 és a i el punt de consulta més proper de p3 és a. Com que volem obtenir 1 facilitat, aquella que fa màxima la mínima distància als punts de consulta, agafem la que té una distància major, que tal i com podem veure a la figura següent és p2.

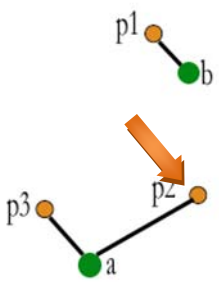


Figura 2.29: Problema Ak-NN: MaxMin

- **MaxSum**. Es tracta de trobar les k facilitats que fan màxima la suma de distàncies als punts de consulta. S'ha de calcular per a cada facilitat la suma de distàncies als punts de consulta i posteriorment obtenir les k facilitats que fan màxima la suma de distàncies.

  Fem el càlcul per a l'exemple de la *figura 2.26*. Si sumem les distàncies de p1 als dos punts de consulta tenim un valor de 4'3, si sumem les distàncies de p2 als dos punts de consulta tenim un valor de 3'8 i si sumem les distàncies de p3 als dos punts de consulta tenim un valor de 4'1. La facilitat amb la suma màxima (tal i com es pot observar en les següents imatges), i per tant, la facilitat resultant és p1.

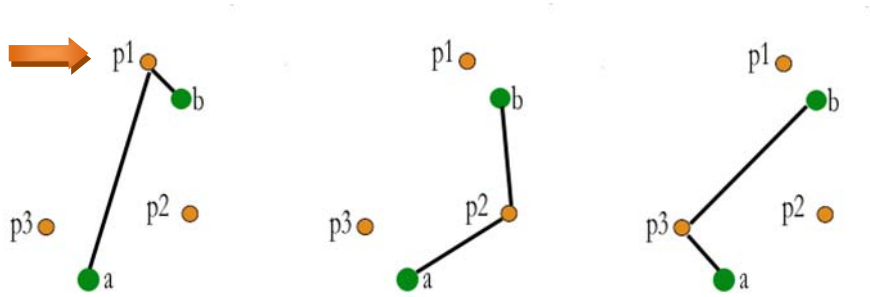


Figura 2.30: Problema Ak-NN: MaxSum

2.1.12. Problema de proximitat Rk-NN

La resolució del problema de proximitat Rk-NN (Reverse k-Nearest Neighbor) o k veïns més propers inversos ens permet trobar els k objectes que es troben més a prop d'un determinat punt que dels altres i donar resposta a preguntar com aquestes: Quins són els 5 edificis que estan més a prop del meu restaurant que dels altres restaurants?

Es pot resoldre el problema Rk-NN a partir del càlcul de cercles i la resolució del problema k-NN explicats anteriorment. El valor k en aquest problema ens indica el grau de proximitat alhora de resoldre k-NN, és a dir, si k=1 haurem de calcular el veí més proper, si k=2 haurem de calcular el segon veí més proper, etc.

Per tal d'entendre més ràpidament el concepte introduirem els Rk-NN sobre el pla, encara que nosaltres després ho apliquem sobre la xarxa de carreteres.

Imaginem que tenim els punts p1, p2, p3 i p4 de la *figura 2.31* i que volem resoldre el problema per a k=1. Per a cada punt calculem quin és el seu veí més proper i fem un cercle amb centre el punt i radi la distància fins al seu veí més proper. Els reversos d'un determinat punt seran tots aquells que dins el seu cercle el contenen.

Veiem quins són els reversos de cada punt:

- Reversos de p1: no en té.
- Reversos de p2: p1 i p3.
- Reversos de p3: p2 i p4.
- Reversos de p4: no en té.

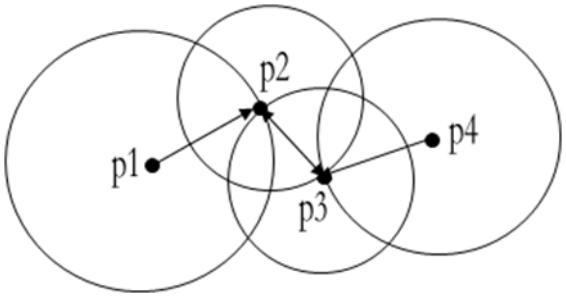


Figura 2.31: Problema Rk-NN

2.2. Conceptes de programació de la GPU

En aquest apartat s'explicaran les nocions fonamentals de la programació de la GPU (Graphics Processing Unit) de les targetes gràfiques modernes. Entendre aquests conceptes i el funcionament de la GPU és necessari per entendre com s'han implementat algunes funcionalitats.

2.2.1. Renderització

El procés de renderització és un procés de càlcul complex que realitza l'ordinador i que té com a finalitat generar una imatge 2D a partir d'una escena 3D, així els objectes geomètrics són processats per pintar-los a pantalla.

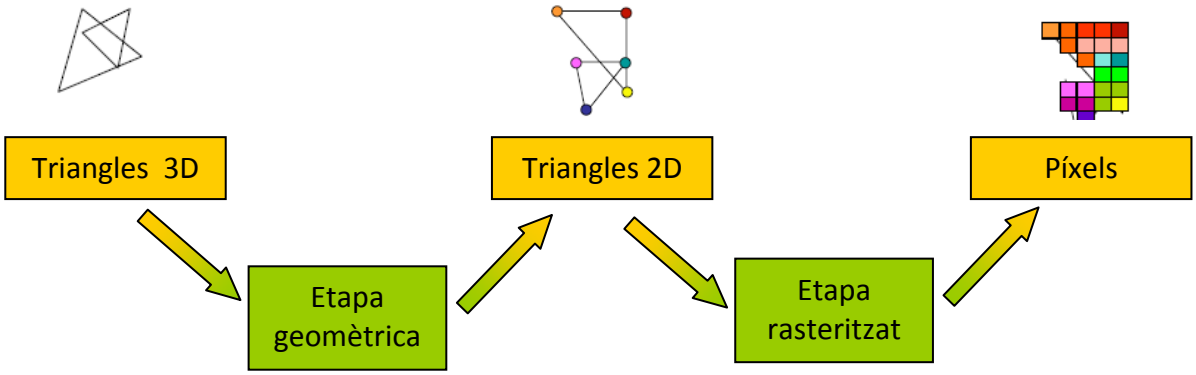


Figura 2.32: Procés simple de renderització

Veient la *figura 2.32* podem fer-nos una idea molt bàsica de com funciona el procés de renderització. A partir de l'escena 3D es passa per l'etapa geomètrica on a partir de les dades d'entrada es tracta cada vèrtex, es busquen les entitats primitives i s'eliminen les primitives geomètriques que queden fora de l'àrea de treball. A partir de l'escena 2D es passa per l'etapa de rasteritzat on es converteix la imatge en píxels i després es tracta cada píxel.

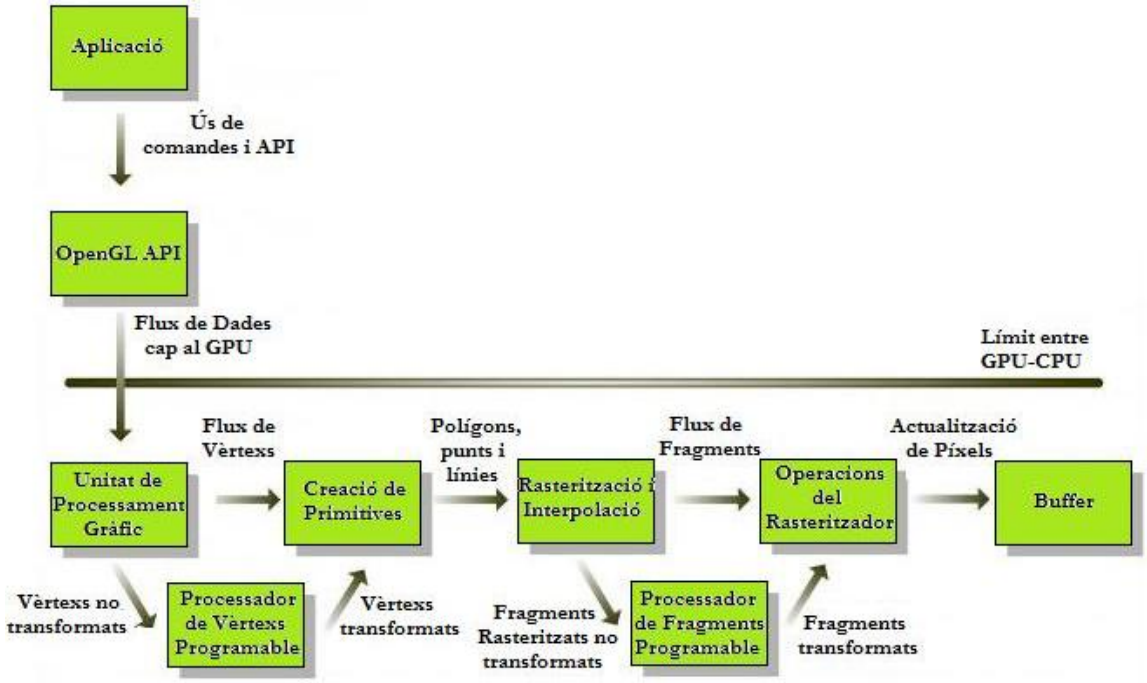


Figura 2.33: Procés de renderització

En la *figura 2.33* podem veure de forma més detallada el procés de renderització. A l'inici del procés, la CPU és l'encarregada d'enviar la geometria i les textures a la GPU per a realitzar les operacions de pintat dels polígons a la pantalla. La GPU s'encarrega de desempaquetar cada vèrtex, i seguidament, el Vèrtex Shader realitza les operacions de transformació i els càlculs d'il·luminació programats pel desenvolupador. Aquestes operacions es realitzen per a cada vèrtex.

A continuació, després de transformar els vèrtexs de coordenades de món a coordenades de pantalla, aquests són combinats per a formar primitives de visualització convertint-se en punts, línies o polígons, segons s'especifiqui. Després de la generació de primitives es realitza el procés de clipping o retallat que consisteix en l'eliminació de part d'una primitiva o grup de primitives que estan fora de la zona de treball.

Posteriorment, les primitives passen a ser rasteritzades convertint-se en píxels amb un color associat. Durant el procés de rasterització es realitza la interpolació de les dades associades a cada vèrtex de la primitiva. Aquests píxels generats per la rasterització i les seves dades interpolades defineix el que es coneix com a fragments.

Un fragment és un píxel candidat a ser pintat a la pantalla. Cada fragment és processat per la unitat programable anomenada Pixel Shader. El color de cada fragment pot ser modificat pel desenvolupador, programant les transformacions desitjades utilitzant les dades interpolades associades al fragment i textures. Un cop finalitzada la transformació del fragment, aquest es pinta en el buffer de renderització (habitualment, la pantalla).

Les noves capacitats de programació unides amb la gran potència de les noves targetes gràfiques proporcionen una gran versatilitat, permetent la implementació de noves tècniques gràfiques amb resultats molt més realistes.

2.2.2. Vèrtex Shader i Pixel Shader

Les targetes gràfiques disposen de dues unitats que permeten incorporar elements programables en les etapes de transformació dels vèrtexs i en la transformació final del color de cada píxel rasteritzat a la pantalla. El procés capaç d'executar les transformacions programables de la geometria s'anomena *Vèrtex Shad*er i el procés capaç d'executar les transformacions de color programables als píxels abans de ser pintats en pantalla s'anomena *Pixel Shader*. Aquest conjunt d'operacions es codifiquen en programes anomenats *shaders*.

Un Vèrtex shader és una funció que rep com a paràmetre un vèrtex. S'executa una vegada per a cada vèrtex i no permet eliminar-lo, tan sols transformar-lo de 3D a coordenades 2D. Es poden manipular propietats com la posició, el color, la textura, etc i es poden aconseguir certs efectes com la deformació en temps real d'un element, per exemple, d'una onada. Té gran importància en el tractament de línies corbes i el seu avenç es veu reflectit en els videojocs més avançats, sobretot en el disseny dels personatges i expressions corporals.

Un pixel shader no intervé en el procés de la definició de l'esquelet de l'escena sinó que forma part de la segona etapa: la rasterització de l'escena. En aquesta part és on s'apliquen les textures i es tracten els píxels que les formen. Un pixel shader, bàsicament, especifica el color i la profunditat d'un píxel permetent que es realitzin càlculs principalment relacionats amb la il·luminació de l'element que formen. La particularitat dels píxels shaders és que, a diferència dels Vèrtex shaders, requereixen un suport de hardware compatible.

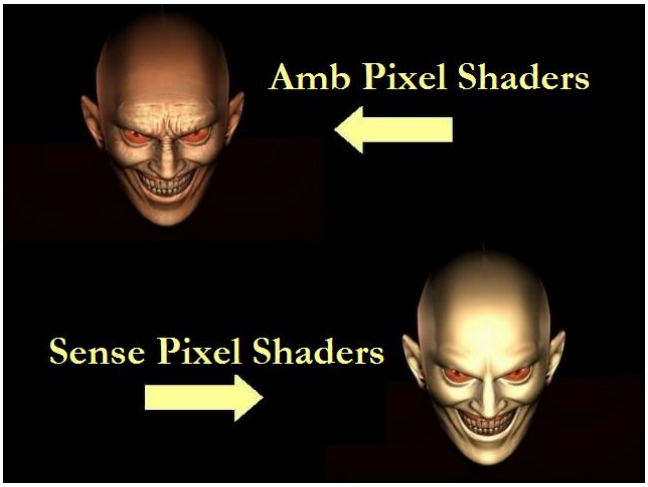


Figura 2.34: Pixel shader

2.2.3. Llenguatges de programació de GPU

Actualment, existeixen varis llenguatges de programació de la GPU. Els més utilitzats són:

- **CG (C for Graphics)**. És un llenguatge compatible amb les dues API[1] gràfiques més importants: OpenGL i DirectX. Ha estat desenvolupat per NVIDIA amb col·laboració amb Microsoft. És un llenguatge similar al llenguatge de programació C, amb la mateixa sintaxi per a les declaracions, les crides a funcions i la majoria dels tipus de dades. No obstant, té algunes millores i modificacions per a facilitar la codificació de programes que es compilin en codi assemblador de GPU optimitzat.

- **GLSL (OpenGL Shading Language)**. És el llenguatge de programació estàndard de la API OpenGL, i al igual que CG, es basa en el llenguatge C. Ha estat desenvolupat per OpenGL ARB.

- **HLSL**. És el llenguatge de programació estàndard de la API Direct3D desenvolupant per Microsoft. Està basat també en el llenguatge C i és gairebé idèntic al CG, degut a que Microsoft i NVIDIA col·laboren conjuntament per al desenvolupament de CG.

Aquests llenguatges de programació de GPU tenen petites diferències entre ells però tots presenten aproximadament el mateix nivell de funcionalitats, no existeixen grans avantatges o desavantatges.

Per al desenvolupament del projecte s'ha decidit treballar amb el llenguatge CG ja que sembla el llenguatge més extés i més compatible.

2.2.4. Llenguatge CG

Gran part de les aplicacions de software actuals estan desenvolupades amb el llenguatge de programació C o C++, però per a crear efectes visuals complexos, els desenvolupadors tenien que utilitzar un llenguatge assemblador que presentava moltes limitacions. Ara, amb el llenguatge Cg es poden crear efectes de qualitat similar a les del cinema i renderitzar-los en temps real en el PC. Ja no és necessari escriure programes directament per al hardware de gràfics, fet que facilita el desenvolupament de sombrejats i efectes visuals en temps real molt sofisticats per entorns OpenGL i DirectX.

[1] API: interfície de programación d'aplicacions. És el conjunt de funcions i procediments que ofereix una biblioteca per a ser utilitzats per un altre software com una capa d'abstracció.

Cg és un llenguatge de programació basat en estàndards i dissenyat per NVIDIA amb col·laboració amb Microsoft per assegurar la compatibilitat amb DirectX i HLSL. Permet aprofitar els grans avantatges experimentats pels processadors gràfics programables i els desenvolupadors poden reutilitzar efectes especials en múltiples plataformes com consoles de jocs, PC i Macintosh.

El llenguatge Cg permet als desenvolupadors realitzar optimitzacions integrades, augmentar l'eficacía dels programes i desenvolupar ràpidament en temps real sombrejat per píxel i per vèrtex amb la finalitat d'obtenir efectes visuals impresionants. El compilador de Cg revoluciona els efectes gràfics generats per ordinador i proporciona a l'escriptori un realisme cinematogràfic. Així mateix, augmenta la productivitat dels programadors i redueix el temps de desenvolupament.

2.3. Mapejat de textures

Fins ara les primitives es dibuixaven en OpenGL amb un sol color o interpolant varis colors en els vèrtexs d'una primitiva. El mapejat de textures descriu el procés de mapeig d'una imatge bidimensional en una primitiva geomètrica o conjunt de primitives.

Per tal de mapejar una textura cal associar els punts d'aquesta amb els punts d'una primitiva geomètrica. Aquesta correspondència es fa utilitzant les coordenades de textura.

En OpenGL, si volem aplicar una textura hem de seguir els següents passos:

- Activar el mapejat de textures.
- Especificar la imatge que s'utilitzarà com a textura.
- Definir el paràmetres d'aplicació de la textura.
- Pintar i assignar les coordenades de textura.

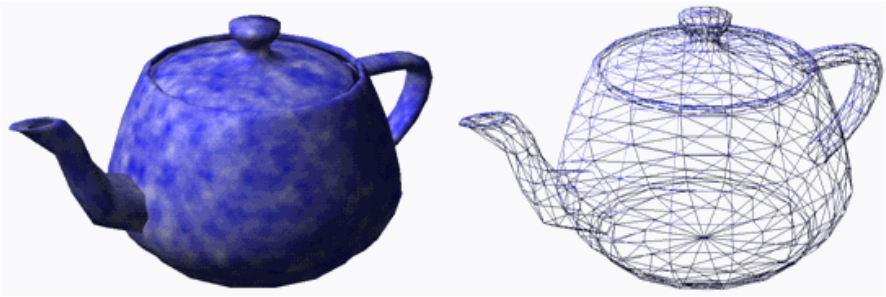


Figura 2.35: Exemple mapejat textura

3. Requeriments

En aquest capítol seran definits els *Requeriments del programari*, els quals recullen, a grans trets, els objectius de l'aplicació juntament amb les seves funcionalitats desitjades. Aquest document ens ha de permetre entendre els elements que envoltaran el sistema informàtic que s'intenta construir, com són: les persones, els procediments, el hardware i el propi software.

Dins d'una aplicació apareixen bàsicament dos tipus de requeriments:

- **Funcionals:** Descriuen quins són els serveis que ens oferirà l'aplicació independentment de la implementació.
- **No Funcionals:** Ens informen sobre les restriccions que venen imposades pel client o pel propi problema.

3.1. Requeriments funcionals

3.1.1. Requeriments generals del sistema

Com a requeriments generals del sistema es mostraran les funcionalitats de les quals disposa l'aplicació reflectint d'aquesta manera les responsabilitats del programa a construir. A continuació es descriuen breument cadascun dels requisits a complir:

- Carregar xarxes de carreteres des d'un fitxer que haurà de tenir l'extensió i el format TIGER/Lines.
- Poder visualitzar la xarxa de carreteres i treballar-hi de forma senzilla permetent realitzar desplaçaments i zooms.
- Afegir objectes a la xarxa de carreteres. Tenim tres tipus d'objectes diferents:
  - Les facilitats, que representen els edificis.
  - Els punts de consulta, que representen les persones.
  - Els vehicles, que representen vehicles.

  Aquests objectes s'han de poder afegir de dues maneres diferents:
  - Introduint-los un a un, indicant l'objecte, les dades i la posició on el volem col·locar.
  - Introduint-los simultàniament de forma aleatòria.
- Poder visualitzar cadascun dels objectes allà on estan situats. Ha d'haver-hi la opció de mostrar només els tipus desitjats.

- Seleccionar objectes de la xarxa per tal de veure'n la seva informació i poder-los identificar fàcilment.

- Eliminar objectes creats a la xarxa de carreteres, ja siguin facilitats, punts de consulta o vehicles. Es podran esborrar un a un seleccionant-los a la pantalla.

- Guardar objectes creats a la xarxa de carreteres en un fitxer per tal de poder-los carregar, si així es desitja, si posteriorment tornem a obrir la mateixa xarxa de carreteres. Aquest fitxer tindrà una extensió i un format determinat.

- Guardar la imatge corresponent a la xarxa de carreteres en un moment determinat. Es tracta d'una captura de pantalla de la finestra actual.

- Generar una estructura de dades per tal d'ordenar els vèrtexs i poder accedir-hi ràpidament (QuadTree de vèrtexs).

- Generar una estructura de dades per tal d'ordenar les arestes i poder accedir-hi ràpidament. (RTree).

- Generar una estructura de dades per tal d'ordenar els objectes estàtics (facilitats i punts de consulta) i accedir-hi ràpidament (QuadTree d'objectes).

- Poder visualitzar cadascuna de les estructures de dades sobre la xarxa de carreteres per veure com estan formades.

- Poder seleccionar una zona de la xarxa (sub-xarxa) i permetre resoldre problemes amb aquesta zona tenint en compte que cal conservar els objectes de la xarxa original. També cal poder retornar a la xarxa original guardant els canvis realitzats.

- Calcular la parametrització de la xarxa de carreteres, és a dir, compactar la xarxa per tal de poder fer els càlculs d'una manera més eficaç i precisa.

- Calcular camí més curt des d'un punt de la xarxa a un o varis nodes utilitzant l'algorisme de camins mínims de Dijkstra.

- Calcular la funció camp de distàncies, a partir d'una facilitat o d'un punt de consulta, que ens servirà per a resoldre els problemes de proximitat.

- Calcular un cercle amb centre a un punt de la xarxa i un radi introduït per l'usuari indicant aquelles facilitats que queden dins la distància indicada, és a dir, dins el cercle.

- Assignar a cada facilitat un color diferent per tal de poder calcular el diagrama de Voronoi i posteriorment resoldre alguns dels problemes de proximitat.

- Calcular els diferents diagrames de Voronoi: proper, llunyà i odre k que ens permetran resoldre els problemes de proximitat.

- Calcular punts d'interès respecte a les facilitats per tal de trobar punts estratègics sobre la xarxa. Els diferents punts són: 1-Center, 1-Center obnoxious, 1-Median i 1-Median obnoxious. Un exemple de problema que podem resoldre amb aquests punts seria trobar el lloc on construir un hospital tenint en compte la localització de les àrees més populars de la ciutat.

- Calcular k veïns més pròxims (k-NN) per tal de trobar les k facilitats que es troben més a prop d'un punt de consulta. Un exemple de problema a resoldre seria trobar els tres hotels més pròxims a la meva posició.

- Calcular k veïns més pròxims inversos (R-k-NN) per tal de trobar les k facilitats que es troben més a prop d'un determinat edifici que dels altres. Un exemple de problema a resoldre seria trobar els cinc edificis que estan més a prop del meu restaurant que dels altres restaurants.

- Calcular k veïns més pròxims agregats (A-k-NN) per tal de trobar les k facilitats millor situades utilitzant una funció de distàncies agregades respecte dels punts de consulta. Les funcions de distància poden ser: MinMax, MinSum, MaxMin, MaxSum. Un exemple de problema a resoldre seria trobar un restaurant que permeti a cinc amics trobar-se recorrent cada un d'ells la mínima distància possible.

- Una interfície gràfica que ens permeti realitzar intuïtivament totes les funcionalitats anteriors i visualitzar la xarxa de carreteres, els objectes creats en aquesta xarxa i els resultats de cada una de les accions que es porten a terme.

3.1.2. Identificació dels actors

A continuació cal identificar els actors de l'aplicació. Un actor és una entitat externa (persona, dispositiu, subsistema, ...) que interactua amb el sistema interpretant un determinat rol.

A la nostra aplicació no hi ha cap mena de manteniment ja que es tracta d'un programa en el qual no apareix distinció entre els possibles usuaris que el puguin utilitzar. Per tant, podem concloure l'existència d'un únic actor, l'usuari que interactua en tot moment amb el sistema.

Com es pot veure, l'esquema mostrat a la *figura 3.1* es força senzill ja que tots els usuaris que utilitzin l'aplicació disposen dels mateixos privilegis.

Figura 3.1: Actors que interactuen amb el sistema

3.1.3. Diagrama de casos d'ús general

Cadascun dels requeriments esmentats anteriorment formarà un cas d'ús ja que descriuen el comportament o funcionalitat del sistema quan aquest interactua amb un usuari extern o actor.

En aquest apartat es mostra mitjançant un diagrama de casos d'ús les relacions existents entre cadascuna de les funcionalitats del sistema (*figura 3.2*). Aquest diagrama s'utilitza per modelar d'alguna manera la vista estàtica del sistema.

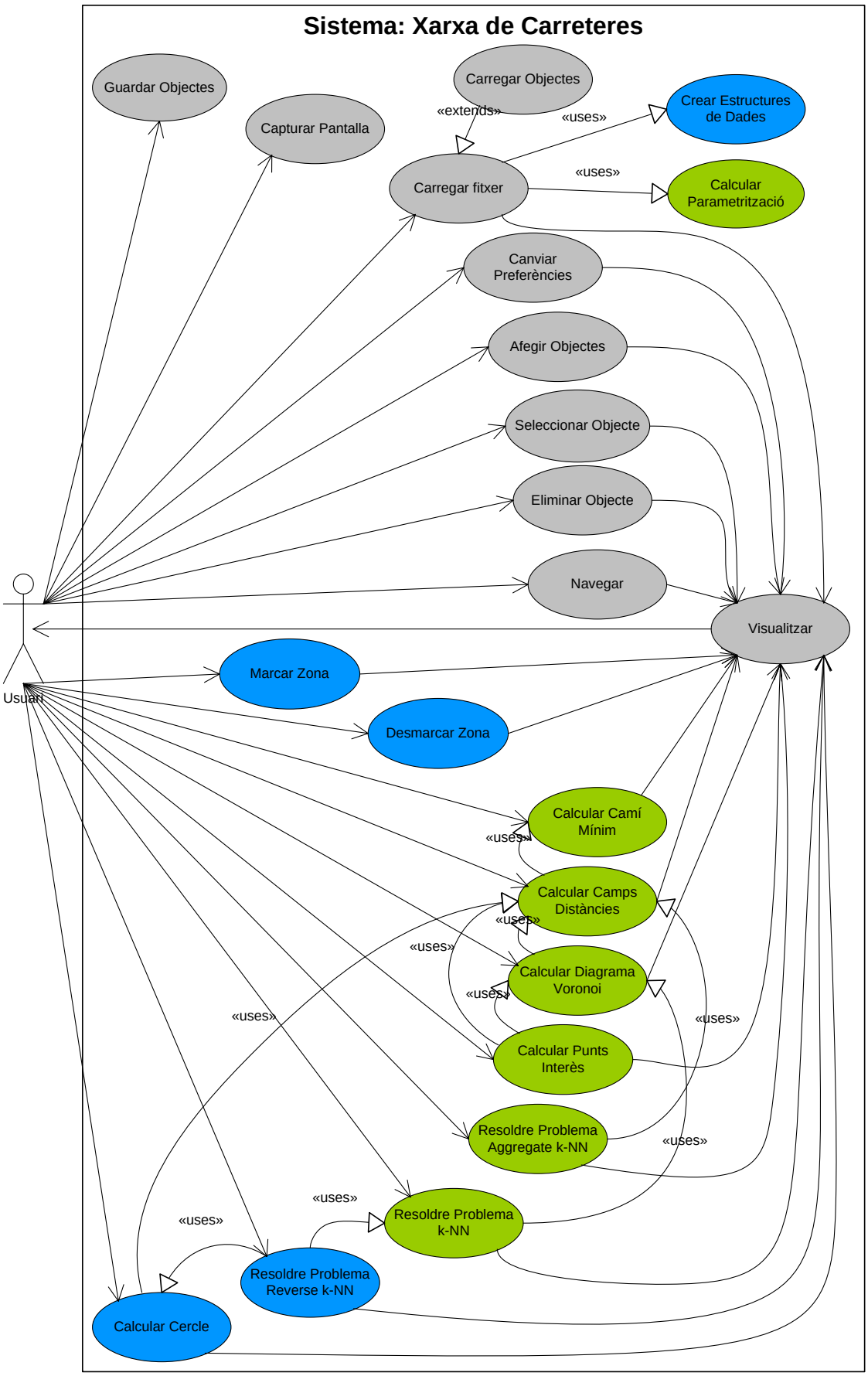
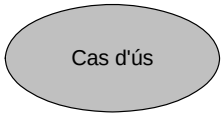
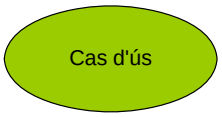


Figura 3.2: Diagrama de casos d'ús general

Tal i com es pot comprovar, els casos d'ús es divideixen bàsicament en tres grans grups:



Funcionalitats comunes (gris): Aquests casos d'ús permeten una interacció bàsica amb l'aplicació. Així doncs, un usuari pot carregar una xarxa de carreteres i afegir i eliminar objectes a través d'una interfície que permet visualitzar-ho i interactuar d'una manera intuïtiva. Altres funcionalitats com la captura de pantalla o el fet de poder canviar les preferències al gust de l'usuari s'han incorporat com a millora i enriquiment de l'aplicació.



Funcionalitats específiques projecte Anna (verd): Trobem diferents casos d'ús. En primer lloc el cas d'ús 'Calcular parametrització' permet compactar la xarxa i poder-hi treballar de forma més precisa i còmode. Seguidament, els casos d'ús 'Calcular camí mínim', 'Calcular camps distàncies' i 'Calcular diagrama Voronoi' s'utilitzen per a fer diferents càlculs sobre la xarxa que posteriorment ens serviran per a resoldre els problemes de proximitat que plantegen els casos d'ús 'Calcular punts d'interès', 'k-NN' i 'Aggregate k-NN'.

Funcionalitats específiques projecte Joan (blau): Trobem diferents casos d'ús. El cas d'ús 'Crear estructures de dades' permet crear i utilitzar les diferents estructures (QuadTree i RTree) per ordenar els diferents objectes de la xarxa. Els casos d'ús 'Marcar zona' i 'Desmarcar zona' permeten crear i eliminar una sub-xarxa respectivament. Finalment, la funcionalitat 'Calcular cercle' ens servirà per a resoldre el problema de proximitat del cas d'ús 'Reverse k-NN'.

Finalment cal remarcar que, tal i com es veu en el diagrama, existeix una gran coherència entre el manteniment de la xarxa i la seva visualització. També es pot apreciar una clara estructuració entre les diferents funcionalitats del sistema.

Les funcionalitats exposades en el diagrama de casos d'ús s'han intentat compactar i simplificar per facilitar-ne la seva comprensió tot i les múltiples relacions que existeixen entre elles. Al següent capítol (anàlisi del sistema) s'exposaran algunes de les funcionalitats més complexes de forma detallada.

3.1.4. Detalls dels requeriments del sistema

Per tal de conèixer amb més exactitud la funcionalitat dels casos d'ús es mostra, per a cada cas d'ús, una fitxa en la qual es descriuran tant l'escenari principal com els escenaris alternatius a més a més d'altres aspectes informatius.

3.1.4.1. Cas d'ús: Carregar Fitxer

Cas d'ús: CARREGAR FITXER	
Descripció	Carregar un fitxer de text que conté la informació necessària per generar el graf de la xarxa de carreteres.
Actor	Usuari.
Precondició	El fitxer ha de tenir el format i l'extensió correcte.
Flux Principal	1. Seleccionar fitxer. 2. Comprovar extensió. 3. **Per a cada línia del fitxer**: 3.1. Llegir coordenada del vèrtex. 3.2. Crear vèrtex. 3.3. Llegir número d'arestes adjacents al vèrtex. 3.4. **Per a cada aresta**: 3.4.1. Llegir número vèrtex contigu. 3.4.2. **Si vèrtex contigu creat**: 3.4.2.1. Llegir tipus d'aresta. 3.4.2.2. Crear aresta. 3.4.2.3. Afegir aresta al vèrtex. 3.4.2.4. Afegir aresta al vèrtex contigu. 4. **Si hi ha fitxer d'objectes**: 4.1. Preguntar a l'usuari si el vol carregar 4.2. Si l'usuari vol carregar el fitxer: 4.2.1. Carregar objectes (veure cas d'ús). 5. **Si el graf no és connex**: 5.1. Trobar sub-grafs connexos. 5.2. Seleccionar el graf amb més vèrtexs. 5.3. Eliminar els altres grafs. 6. Crear graf. 7. Crear estructures de dades (veure cas d'ús). 8. Crear parametrització de la xarxa (veure cas d'ús).
Flux Alternatiu	2. Si l'extensió no és correcta mostrar missatge d'error. 3. Si línia incorrecta mostrar missatge d'error.
Postcondició	Xarxa de carreteres creada correctament.
Observacions	- El punt 4.2.1 és una extensió que permet carregar facilitats, punts de consulta i vehicles si anteriorment

	l'usuari els havia guardat. - Per a resoldre els problemes de proximitat és important que el graf sigui connex (hi ha camí entre qualsevol parella de vèrtexs) ja que les parts no connexes dificulten els càlculs i no aporten cap millora. - En el punt 7 es creen les diferents estructures de dades que necessita l'aplicació per poder treballar d'una manera més eficient amb la xarxa de carreteres. - En el punt 8 es crea la parametrització de la xarxa de carreteres per tal de comprimir-la i facilitar-ne els càlculs posteriors.

3.1.4.2. Cas d'ús: Navegar

Cas d'ús: NAVEGAR	
Descripció	Realitzar un desplaçament o un zoom a la xarxa de carreteres. Aquesta funcionalitat és molt útil per a l'usuari per a poder visualitzar zones de la xarxa amb més claredat.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. Escollir opció. 2. **Si realitzar desplaçament**: 2.1. Marcar punt de la pantalla amb el botó esquerre del ratolí. 2.2. **Mentre no es deixi de prémer el botó del ratolí**: 2.2.1. Calcular desplaçament del ratolí a l'eix X. 2.2.2. Calcular desplaçament del ratolí a l'eix Y. 2.2.3. Canviar punt de vista de l'objectiu. 3. **Altrament si canviar zoom**: 3.1. Marcar punt de la pantalla amb el botó dret del ratolí. 3.2. **Mentre no es deixi de prémer el botó del ratolí:** 3.2.1. Calcular desplaçament del ratolí a l'eix Y. 3.2.2. Canviar zoom de l'objectiu.
Flux Alternatiu	
Postcondició	Desplaçament o zoom realitzat correctament.
Observacions	

3.1.4.3. Cas d'ús: Carregar Objectes

Cas d'ús: CARREGAR OBJECTES	
Descripció	Carregar un fitxer de text que conté la informació necessària per generar un conjunt d'objectes, ja siguin facilitats, punts de consulta o vehicles.
Actor	Usuari.
Precondició	El fitxer ha d'existir i ha de tenir el format i l'extensió correcte.
Flux Principal	1. Obrir fitxer. 2. **Per a cada línia del fitxer**: 2.1. Llegir tipus de línia. 2.2. **Si el tipus és facilitat:** 2.2.1. Llegir coordenada de la facilitat. 2.2.2. Llegir aresta que forma part la facilitat. 2.2.3. Llegir punt de parametrització de la facilitat. 2.2.4. Llegir color assignat a la facilitat. 2.2.5. Llegir nom, adreça i web de la facilitat. 2.2.6. Llegir tipus de la facilitat. 2.2.7. Crear facilitat. 2.2.8. Afegir la facilitat a la llista. 2.3. **Altrament si el tipus és punt de consulta**: 2.3.1. Llegir coordenada del punt de consulta. 2.3.2. Llegir aresta que forma part el punt de consulta. 2.3.3. Llegir punt de parametrització del punt de consulta. 2.3.4. Llegir color assignat al punt de consulta. 2.3.5. Llegir nom del punt de consulta. 2.3.6. Crear punt de consulta. 2.3.7. Afegir el punt de consulta a la llista. 2.4. **Altrament si el tipus és car**: 2.4.1. Llegir coordenada del car. 2.4.2. Llegir aresta que forma part el car. 2.4.3. Llegir el sentit del vehicle (vèrtex destí). 2.4.4. Llegir punt de parametrització del car. 2.4.5. Llegir color assignat al car. 2.4.6. Llegir nom del car. 2.4.7. Llegir velocitat del car. 2.4.8. Crear car. 2.4.9. Afegir vehicle a la llista.
Flux Alternatiu	2. Si línia incorrecta mostrar missatge d'error.
Postcondició	Conjunt d'objectes creats correctament.
Observacions	- L'atribut de color que es llegeix dels objectes no es

	refereix al color que s'utilitza per pintar-la sobre la xarxa de carreteres, sinó el color que s'utilitza per a resoldre problemes de proximitat.

3.1.4.4. Cas d'ús: Guardar Objectes

Cas d'ús: GUARDAR OBJECTES	
Descripció	Guardar en un fitxer de text la informació de tots els objectes que hi hagi a la xarxa perquè posteriorment es puguin carregar.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. Crear fitxer. 2. **Per a cada facilitat de la xarxa**: 2.1. Escriure que el tipus de línia és facilitat 2.2. Escriure coordenada de la facilitat. 2.3. Escriure aresta que forma part la facilitat. 2.4. Escriure punt de parametrització de la facilitat. 2.5. Escriure color assignat a la facilitat. 2.6. Escriure nom, adreça i web de la facilitat. 2.7. Escriure tipus de la facilitat. 3. **Per a cada punt de consulta de la xarxa**: 3.1. Escriure que el tipus de línia és punt de consulta. 3.2. Escriure coordenada del punt de consulta. 3.3. Escriure aresta que forma part el punt de consulta. 3.4. Escriure punt de parametrització del punt de consulta. 3.5. Escriure color assignat al punt de consulta. 3.6. Escriure nom del punt de consulta. 4. **Si el tipus és car**: 4.1. Escriure que el tipus de línia és car. 4.2. Escriure coordenada del car. 4.3. Escriure aresta que forma part el car. 4.4. Escriure el sentit del vehicle (vèrtex destí). 4.5. Escriure punt de parametrització del car. 4.6. Escriure color assignat al car. 4.7. Escriure nom del car. 4.8. Escriure velocitat del car.
Flux Alternatiu	
Postcondició	Objectes guardats en un fitxer de text correctament.
Observacions	- En el punt 1, quan es crea el fitxer d'objectes se li assigna el mateix nom que el fitxer de la xarxa de carreteres, però canviant-li la extensió.

	- Per a guardar els objectes en el fitxer, el que es fa és utilitzar una línia per a cada una d'elles, on el primer caràcter de la línia ens informa de si hem de llegir una facilitat, un punt de consulta o un car. - Igual que en el cas d'ús anterior, l'atribut de color que s'escriu dels objectes no es refereix al color que s'utilitza per pintar-los sobre la xarxa de carreteres, sinó el color que s'utilitza per a resoldre problemes de proximitat.

3.1.4.5. Cas d'ús: Afegir Objectes

Cas d'ús: AFEGIR OBJECTES	
Descripció	Afegir facilitats, punts de consulta i/o vehicles a la xarxa de carreteres. Existeixen dos maneres de fer-ho: afegir-les una a una introduint la seva ubicació i les seves dades o afegir-ne vàries a ubicacions aleatòries.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. Escollir opció. 2. **Si afegir facilitat**: 2.1. Seleccionar punt sobre la xarxa de carreteres. 2.2. Buscar aresta més proper al punt utilitzant l'RTree. 2.3. Buscar coordenada de l'aresta més pròxima al punt. 2.4. Introduir nom, adreça i web de la facilitat. 2.5. Introduir tipus de facilitat. 2.6. Calcular punt de parametrització. 2.7. Crear facilitat. 2.8. Afegir facilitat a la xarxa de carreteres utilitzant el QuadTree d'objectes. 3. **Altrament si afegir punt de consulta**: 3.1. Seleccionar punt sobre la xarxa de carreteres. 3.2. Buscar aresta més proper al punt utilitzant l'RTree. 3.3. Buscar coordenada de l'aresta més pròxima al punt. 3.4. Introduir nom del punt de consulta. 3.5. Calcular punt de parametrització. 3.6. Crear punt de consulta. 3.7. Afegir punt de consulta a la xarxa de carreteres utilitzant el QuadTree d'objectes. 4. **Altrament si afegir car**: 4.1. Seleccionar punt sobre la xarxa de carreteres. 4.2. Buscar aresta més proper al punt utilitzant l'RTree. 4.3. Buscar coordenada de l'aresta més pròxima al punt.

	4.4. Introduir nom del car. 4.5. Introduir velocitat del car. 4.6. Calcular punt de parametrització. 4.7. Crear car. 4.8. Afegir vehicle a la xarxa de carreteres. 5. **Altrament si afegir objectes aleatoris**: 5.1. **Si afegir facilitats:** 5.1.1. Introduir nombre. 5.1.2. Introduir tipus específic o aleatori. 5.1.3. **Per a cada facilitat a afegir**: 5.1.3.1. Obtenir aresta aleatori de l'RTree. 5.1.3.2. Obtenir coordenada aleatòria de l'aresta. 5.1.3.3. Assignar tipus (específic o aleatori). 5.1.3.4. Calcular punt parametrització. 5.1.3.5. Crear facilitat. 5.1.3.6. Afegir facilitat a la xarxa de carreteres utilitzant el QuadTree d'objectes. 5.2. **Si afegir punts de consulta:** 5.2.1. Introduir nombre. 5.2.2. **Per a cada punt de consulta a afegir**: 5.2.2.1. Obtenir aresta aleatori de l'RTree. 5.2.2.2. Obtenir coordenada aleatòria de l'aresta. 5.2.2.3. Calcular punt parametrització. 5.2.2.4. Crear punt de consulta. 5.2.2.5. Afegir punt de consulta a la xarxa de carreteres utilitzant el QuadTree d'objectes. 5.3. **Si afegir vehicles:** 5.3.1. Introduir nombre. 5.3.2. Introduir velocitat específica o aleatòria. 5.3.3. **Per a cada vehicle a afegir**: 5.3.3.1. Obtenir aresta aleatori de l'RTree. 5.3.3.2. Obtenir coordenada aleatòria de l'aresta. 5.3.3.3. Assignar velocitat (específic o aleatori). 5.3.3.4. Calcular punt parametrització. 5.3.3.5. Crear car. 5.3.3.6. Afegir facilitat a la xarxa de carreteres.
Flux Alternatiu	2.2. Si no s'ha pogut trobar cap aresta a partir del punt donat per l'usuari, no es continuarà afegint la facilitat i l'usuari podrà tornar a marcar el punt. 3.2. Si no s'ha pogut trobar cap aresta a partir del punt donat per l'usuari, no es continuarà afegint el punt de consulta i l'usuari podrà tornar a marcar el punt.

	4.2. Si no s'ha pogut trobar cap aresta a partir del punt donat per l'usuari, no es continuarà afegint el vehicle i l'usuari podrà tornar a marcar el punt.
Postcondició	Un o varis objectes creats correctament.
Observacions	- En els punt 2.2, 3.2 i 4.2 s'utilitza l'estructura de dades RTree per a trobar si hi ha una aresta pròxim al punt introduït per l'usuari. - En els punts 2.3, 3.3 i 4.3 el que es fa és, a partir de l'aresta seleccionat i el punt donat per l'usuari, buscar el punt sobre l'aresta que estigui més pròxim al punt donat per l'usuari. D'aquesta manera s'aconsegueix ubicar l'objecte exactament sobre l'aresta evitant possibles errors. - En els punts 2.8 i 3.7 s'afegeix l'objecte a la xarxa de carreteres. Per fer-ho, s'introdueix dins la estructura de dades QuadTree d'objectes. Cal recordar que els vehicles no formen part d'aquest QuadTree, ja que a causa del seu constant moviment no es poden ordenar. - En el punt 5.1.2 s'escull el tipus de les facilitats que es crearan. L'usuari té la opció de crear totes les facilitats d'un mateix tipus o indicar que aquest sigui aleatori i així afegir tot tipus de facilitats. - En el punt 5.3.2 s'escull el tipus de velocitat del vehicle que es crearà. Igual que la observació anterior, l'usuari podrà assignar la mateixa velocitat a tots els vehicles o generar-les de manera aleatòria.

3.1.4.6. Cas d'ús: Eliminar Objecte

Cas d'ús: ELIMINAR OBJECTE	
Descripció	Eliminar un objecte de la xarxa de carreteres, ja sigui una facilitat, un punt de consulta o un car.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada i existeix almenys un objecte.
Flux Principal	1. Seleccionar punt sobre la xarxa de carreteres. 2. Buscar objecte (facilitat o punt de consulta) propera al punt utilitzant el QuadTree d'objectes. 3. Buscar objecte (car) propera al punt. 4. Calcular quin objecte (facilitat, punt de consulta o car) es troba més a prop del punt entrat per l'usuari. 5. **Si l'objecte és una facilitat o punt de consulta**: 5.1. Eliminar objecte del QuadTree d'objectes.

	6. **Altrament si l'objecte és un car:** 6.1. Eliminar objecte de la llista de vehicles.
Flux Alternatiu	2 i 3. Si entre els dos apartats no s'ha pogut trobar cap objecte, no es continuarà eliminant l'objecte i l'usuari podrà tornar a marcar el punt.
Postcondició	Objecte eliminat correctament.
Observacions	- En el punt 2 es busca l'objecte més pròxim al punt donat per l'usuari utilitzant el QuadTree d'objectes. Com sabem, aquest QuadTree només guarda les facilitats i els punts de consulta de la xarxa (els vehicles no, ja que degut al seu constant moviment, no els podem ordenar), per tant només retornarà la facilitat o punt de consulta pròxima al punt. - En el punt 3 es busca l'objecte (car) pròxim al punt donat per l'usuari. En aquest cas no s'utilitza el QuadTree, sinó que s'haurà de fer un recorregut per a totes elles.

3.1.4.7. Cas d'ús: Seleccionar Objecte

Cas d'ús: SELECCIONAR OBJECTE	
Descripció	Seleccionar un objecte de la xarxa de carreteres, ja sigui una facilitat, un punt de consulta o un car, i mostrar la seva informació.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada i existeix almenys un objecte.
Flux Principal	1. Seleccionar punt sobre la xarxa de carreteres. 2. Buscar objecte (facilitat o punt de consulta) propera al punt utilitzant el QuadTree d'objectes. 3. Buscar objecte (car) propera al punt. 4. Calcular quin objecte (facilitat, punt de consulta o car) es troba més a prop del punt entrat per l'usuari. 5. **Si l'objecte és una facilitat**: 5.1. Obrir quadre de text al costat de la facilitat. 5.2. Mostrar nom, adreça i web de la facilitat. 5.3. Mostrar tipus de la facilitat. 6. **Altrament si l'objecte és un punt de consulta:**

	6.1. Obrir quadre de text al costat del punt de consulta. 6.2. Mostrar nom del punt de consulta. 7. **Altrament si l'objecte és un car:** 7.1. Obrir quadre de text al costat del car. 7.2. Mostrar nom del car. 7.3. Mostrar velocitat del car.
Flux Alternatiu	2 i 3. Si entre els dos apartats no s'ha pogut trobar cap objecte, no es continuarà seleccionant l'objecte i l'usuari podrà tornar a marcar el punt.
Postcondició	Quadre de text mostrant la informació de l'objecte mostrat correctament.
Observacions	- En el punt 2 es busca l'objecte més pròxim al punt donat per l'usuari utilitzant el QuadTree d'objectes. Com sabem, aquest QuadTree només guarda les facilitats i els punts de consulta de la xarxa (els vehicles no, ja que degut al seu constant moviment, no els podem ordenar), per tant només retornarà la facilitat o punt de consulta pròxima al punt. - En el punt 3 es busca l'objecte (car) pròxim al punt donat per l'usuari. En aquest cas no s'utilitza el QuadTree, sinó que s'haurà de fer un recorregut per a totes elles.

3.1.4.8. Cas d'ús: Capturar Pantalla

Cas d'ús: CAPTURAR PANTALLA	
Descripció	Realitzar una captura de pantalla i guardar-la en fitxer d'imatge. Aquesta captura es realitza sobre la part de la interfície que conté la xarxa de carreteres, deixant de banda la part dels menús i barra d'eines.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. Seleccionar fitxer on volem guardar la imatge. 2. Seleccionar format d'imatge. 3. Comprovar extensió. 4. Guardar xarxa de carreteres.
Flux Alternatiu	3. Si la extensió és diferent a .JPG o .BMP li afegirà automàticament.
Postcondició	Xarxa de carreteres guardada en un fitxer d'imatge correctament.
Observacions	- En el punt 2 es pot seleccionar entre els formats d'imatge JPG i BMP.

3.1.4.9. Cas d'ús: Canviar Preferències

Cas d'ús: CANVIAR PREFERÈNCIES	
Descripció	Canviar les preferències de l'aplicació per a millorar-ne el rendiment. Cada xarxa de carreteres que obrim a la nostra aplicació tindrà les seves preferències, ja que depenent del número de vèrtexs de cada una d'elles necessitarem un millor rendiment.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. Mostrar preferències actuals. 2. Activar o desactivar visualització del mapa detallada. 3. Activar o desactivar ajuda a la selecció d'objectes. 4. Desar canvis.
Flux Alternatiu	
Postcondició	Preferències de la xarxa de carreteres canviades correctament.
Observacions	- La opció que l'usuari pot activar o desactivar en el punt 2 s'anomena visualització del mapa detallada. Aquesta serveix per a poder veure les carreteres de la xarxa d'una manera més detallada (amb gruixos) o sense detall (una línia). Desactivar aquesta opció millora bastant el rendiment de l'aplicació quan visualitzem el mapa, mentre que deixar-la activada equival a visualitzar la xarxa d'una manera més real. L'usuari podrà jugar amb aquest factor donant prioritat al rendiment o a la visualització. - Per altra banda, la opció del punt 3 s'anomena ajuda a la selecció d'objectes. Això succeeix quan l'usuari vol seleccionar un punt d'una aresta o vol seleccionar un objecte. Si l'usuari decideix activar aquesta opció, quan mogui el ratolí per damunt dels objectes, aquests s'il·luminaran automàticament. Per fer-ho, l'aplicació haurà d'anar calculant sobre quin punt es troba l'usuari en tot moment. Així doncs, igual que en el cas anterior, quan l'usuari es trobi en una xarxa de carreteres molt gran haurà de desactivar aquesta opció.

3.1.4.10. Cas d'ús: Calcular Parametrització de la Xarxa

Cas d'ús: CALCULAR PARAMETRITZACIÓ DE LA XARXA	
Descripció	Calcular la parametrització de la xarxa de carreteres. Hem creat una parametrització on cada aresta es representa una sola vegada com un segment col·locat en una reixa rectangular.
Actor	Usuari.
Precondició	El graf de la xarxa s'ha d'haver creat correctament.
Flux Principal	1. Calcular la longitud total d'arestes. 2. Calcular el nombre de píxels que representarà cada unitat de longitud depenent del nombre de píxels totals que tinguem per a calcular la parametrització. 3. **Per a cada aresta**: 3.1. Calcular nombre de píxels que li corresponen. 3.2. Buscar una fila amb prou espai per a l'aresta. 3.3. Calcular punt inici parametrització. 3.4. Calcular punt final parametrització. 3.5. Guardar punts de parametrització a l'aresta.
Flux Alternatiu	3.1. El nombre mínim de píxels serà 1.
Postcondició	Parametrització de la xarxa creada correctament.
Observacions	

3.1.4.11. Cas d'ús: Calcular Camí Mínim

Cas d'ús: CALCULAR CAMÍ MÍNIM	
Descripció	Es tracta de calcular el camí mínim des d'un punt de consulta fins a una facilitat o des d'una facilitat fins a tots els vèrtexs. Per a calcular aquest camí es parteix de la punt de consulta o facilitat inicial i aplicant l'algorisme de camins mínims de Dijkstra es troba el camí mínim per arribar a tots els vèrtexs de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - Ha d'existir almenys un punt de consulta (només si calculem el camí mínim d'un punt de consulta a una facilitat). - Ha d'existir almenys una facilitat.

Flux Principal	1. Seleccionar punt de consulta o facilitat inicial. 2. Inicialitzar cost vèrtexs a -1. 3. Obtenir aresta de la punt de consulta o facilitat inicial. 4. Buscar els dos vèrtexs d'aquesta aresta. 5. Assignar cost als dos vèrtexs, que serà el resultat de calcular la distància entre el punt de consulta i el vèrtex. 6. Indicar que són vèrtexs inicials, que no tenen antecessor. 7. Crear un vector de vèrtexs per col·locar tots aquells vèrtexs que encara no tenen el cost mínim definitiu. 8. Afegir els dos vèrtexs inicials al vector. 9. **Mentre hi hagi vèrtexs al vector**: 9.1. Buscar vèrtex amb menys cost, aquest serà el vèrtex actual. 9.2. **Per a cada aresta que surt del vèrtex**: 9.2.1. Obtenir cost aresta. 9.2.2. Obtenir vèrtex contigu. 9.2.3. Obtenir el cost actual sumant el cost del vèrtex actual i el cost de l'aresta. 9.2.4. **Si el vèrtex contigu té cost -1**: 9.2.4.1. Assignar cost actual al vèrtex contigu. 9.2.4.2. Indicar com a antecessor d'aquest vèrtex el vèrtex actual. 9.2.4.3. Afegir el vèrtex contigu al vector. 9.2.5. **Altrament si el vèrtex contigu té cost superior al cost actual**: 9.2.5.1. Assignar cost actual al vèrtex contigu. 9.2.5.2. Indicar com a antecessor d'aquest vèrtex el vèrtex actual. 9.3. Treure vèrtex actual del vector. 10. **Si calcular camí mínim punt de consulta-facilitat**: 10.1. Obtenir aresta facilitat destí. 10.2. Buscar els dos vèrtexs d'aquest aresta. 10.3. Calcular el vèrtex més proper a la facilitat. 10.4. **Mentre vèrtex no sigui l'inicial**: 10.4.1. Obtenir vèrtex antecessor.
Flux Alternatiu	
Postcondició	Camí mínim calculat correctament.
Observacions	- Sempre es calcula el camí mínim des del punt de consulta o facilitat inicial fins a tots els vèrtexs de la xarxa ja que hi ha algunes funcionalitats que necessiten aquest càlcul. Tot i això, si el que volem és el camí des d'una punt de consulta per arribar a una facilitat determinada, el que es mostrarà a l'usuari és el camí que va del punt de consulta a la facilitat indicada.

	- Per a obtenir el cost s'utilitza la distància entre els vèrtexs.

3.1.4.12. Cas d'ús: Calcular Camp Distàncies

Cas d'ús: CALCULAR CAMP DISTÀNCIES	
Descripció	Calcular la funció camp distàncies. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització s'ha d'haver calculat. - Ha d'existir almenys una facilitat.
Flux Principal	1. Escollir facilitat. 2. Calcular camí mínim d'aquesta facilitat(veure cas d'ús). 3. Sobre la parametrització crear la funció distància aplicant el cost de cada vèrtex obtingut en el camí mínim. 4. **Per a cada píxel**: 4.1. Normalitzar el valor entre [0,1]. 5. Guardar funció camp distàncies.
Flux Alternatiu	
Postcondició	La funció camp distàncies s'ha creat correctament.
Observacions	

3.1.4.13. Cas d'ús: Assignar Color Facilitats

Cas d'ús: ASSIGNAR COLOR FACILITATS	
Descripció	Generar i assignar un color diferent per a cada facilitat.
Actor	Usuari.
Precondició	Ha d'existir almenys una facilitat.
Flux Principal	1. Calcular nombre de facilitats. 2. Crear un vector de valors a utilitzar per a generar els colors. 3. Afegir dos valors per defecte al vector: el valor 0 i el valor 1. 4. A partir del nombre de facilitats que tenim podem calcular quants valors necessitem per a poder generar tots els colors diferents. Utilitzarem la combinatòria per a poder-ho calcular, exactament les variacions amb repetició. 5. **Si necessitem més de dos valors per a generar els colors, per a cada valor de més que necessitem**:

	5.1.1. Generar valor. 5.1.2. Afegir valor al vector. 6. Generar totes les combinacions de colors possibles amb els valors del vector. 7. **Per a cada facilitat**: 7.1. Assignar un dels colors generats anteriorment que encara no hagi estat assignat a cap altra facilitat.
Flux Alternatiu	
Postcondició	A cada facilitat se li ha assignat un color diferent.
Observacions	- Cada color a generar té 3 components RGB (vermell, verd i blau). Cada component pot tenir un valor entre 0 i 1. S'utilitzaran tots els valors que siguin necessaris per tal de generar un color diferent per a cada facilitat. - S'intentaran generar els colors el màxim de diferents possibles, així si per exemple necessitem 3 valors per a generar els colors utilitzarem el 0, el 0.5 i el 1. - No s'utilitzaran els colors blanc i negre.

3.1.4.14. Cas d'ús: Calcular Diagrama Voronoi

Cas d'ús: CALCULAR DIAGRAMA VORONOI	
Descripció	Calcular el diagrama de Voronoi proper, llunyà o d'ordre k. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització de la xarxa s'ha d'haver calculat. - Ha d'existir almenys una facilitat. - Cada facilitat ha de tenir assignat un color.
Flux Principal	1. **Per a cada facilitat**: 1.1. Calcular camp distàncies de la facilitat (veure cas d'ús). 2. Escollir opció. 3. **Si calcular diagrama de Voronoi proper**: 3.1. Per a cada píxel de la parametrització: 3.1.1. Buscar quina és la facilitat que té més a prop. 3.1.2. Pintar píxel amb el color assignat a la facilitat més propera. 4. **Altrament si calcular diagrama de Voronoi llunyà**: 4.1. Per a cada píxel de la parametrització: 4.1.1. Buscar quina és la facilitat que té més lluny. 4.1.2. Pintar píxel amb el color assignat a la facilitat

	més llunyana. 5. **Altrament si calcular diagrama de Voronoi ordre k**: 5.1. Introduir valor k. 5.2. Per a cada una de les k capes: 5.2.1. Per a cada píxel de la parametrització: 5.2.1.1. Buscar quina és la k facilitat més propera. 5.2.1.2. Pintar píxel amb el color amb el color assignat a la k facilitat més propera.
Flux Alternatiu	5.1. El valor de k ha de set com a mínim 1 i com a màxim el nombre de facilitats de la xarxa.
Postcondició	Diagrama de Voronoi calculat correctament.
Observacions	Tot i que amb una sola facilitat ja es pot calcular el diagrama de Voronoi, cal dir que no té cap mena de sentit, com a mínim hauria d'haver-n'hi dues.

3.1.4.15. Cas d'ús: Calcular Punts d'Interès

Cas d'ús: CALCULAR PUNTS D'INTERÈS	
Descripció	Calcular els diferents punts d'interès de la xarxa respecte a les facilitats. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització de la xarxa s'ha d'haver calculat. - Ha d'existir almenys una facilitat.
Flux Principal	1. **Per a cada facilitat**: 1.1. Calcular camp distàncies de la facilitat (veure cas d'ús). 2. **Calcular 1-Center**: 2.1. Calcular diagrama Voronoi llunyà (veure cas d'ús). 2.2. Buscar punt del Voronoi llunyà amb depth inferior. 2.3. Obtenir aresta on es troba aquest punt. 2.4. Calcular coordenada exacta del punt sobre l'aresta. 2.5. Crear punt 1-Center a partir d'aquesta coordenada. 3. **Calcular 1-Center obnoxious**: 3.1. Calcular diagrama Voronoi proper (veure cas d'ús). 3.2. Buscar punt del Voronoi proper amb depth superior. 3.3. Obtenir aresta on es troba aquest punt 3.4. Calcular coordenada exacta del punt sobre l'aresta. 3.5. Crear punt 1-Center obnoxious a partir d'aquesta coordenada.

	4. **Calcular 1-Median o 1-Median obnoxious**: 4.1. **Per a cada facilitat**: 4.1.1. Sumar camp distàncies de la facilitat ponderat sobre el nombre total de facilitats. 4.2. **Calcular 1-Median**: 4.2.1. Buscar punt de la suma de camps amb depth inferior. 4.2.2. Obtenir aresta on es troba aquest punt 4.2.3. Calcular coordenada exacta del punt sobre l'aresta. 4.2.4. Crear punt 1-Median a partir d'aquesta coordenada. 4.3. **Calcular 1-Median obnoxious**: 4.3.1. Buscar punt de la suma de camps amb depth superior. 4.3.2. Obtenir aresta on es troba aquest punt. 4.3.3. Calcular coordenada exacta del punt sobre l'aresta. 4.3.4. Crear punt 1-Median obnoxious a partir d'aquesta coordenada.
Flux Alternatiu	
Postcondició	Punts d'interès calculats correctament.
Observacions	

3.1.4.16. Cas d'ús: Resoldre Problema Aggregate k-NN

Cas d'ús: RESOLDRE PROBLEMA AGGREGATE K-NN	
Descripció	Resoldre un problema del tipus Aggregate k-NN. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa. Podem utilitzar una d'aquestes quatre funcions distància per a fer el càlcul: - MinMax: trobem les k facilitats que fan mínima la màxima distància als punts de consulta. - MinSum: trobem les k facilitats que fan mínima la suma de distàncies als punts de consulta. - MaxMin: trobem les k facilitats que fan màxima la mínima distància als punts de consulta. - MaxSum: trobem les k facilitats que fan màxima la

	suma de distàncies als punts de consulta.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització de la xarxa s'ha d'haver calculat. - Ha d'existir almenys una facilitat. - Ha d'existir almenys un punt de consulta.
Flux Pincipal	1. Introduir tipus de facilitats a utilitzar per a resoldre el problema. 2. Introduir nombre k de facilitats a buscar. 3. Escollir la funció de distància a utilitzar respecte els punts de consulta. 4. **Per a cada facilitat**: 4.1. Calcular camp distàncies facilitat (veure cas d'ús). 5. **Si calcular utilitzant MinMax**: 5.1. Per a cada facilitat: 5.1.1. Calcular la distància a tots els punts de consulta. 5.1.2. Guardar el punt de consulta amb distància major. 5.2. Obtenir les k facilitats que tenen una menor distància guardada, és a dir que tenen el punt de consulta més llunyà a menor distància. 6. **Si calcular utilitzant MaxMin**: 6.1. Per a cada facilitat: 6.1.1. Calcular la distància a tots els punts de consulta. 6.1.2. Guardar el punt de consulta amb distància menor. 6.2. Otenir les k facilitats que tenen una major distància guardada, és a dir, que tenen el punt de consulta més proper a major distància. 7. **Si calcular utilitzant MinSum**: 7.1. Per a cada facilitat: 7.1.1. Calcular la distància a tots els punts de consulta. 7.1.2. Sumar totes les distàncies a punts de consulta. 7.2. Obtenir les k facilitats que tenen la suma de distàncies menor. 8. **Si calcular utilitzant MaxSum**: 8.1. Per a cada facilitat: 8.1.1. Calcular la distància a tots els punts de consulta. 8.1.2. Sumar totes les distàncies a punts de consulta. 8.2. Obtenir les k facilitats que tenen la suma de distàncies major.

Flux Alternatiu	1. Si no hi ha cap facilitat del tipus indicat mostrar un missatge d'avís. 2. El nombre k ha de ser com a mínim 1 i com a màxim el nombre de facilitats del tipus que busquem. Si el nombre k és superior, mostrar missatge d'avís i buscar només les es puguin.
Postcondició	Problema Aggregate k-NN resolt correctament.
Observacions	Si s'indica un tipus concret de facilitat només s'utilitzaran les d'aquells tipus per a fer els càlculs.

3.1.4.17. Cas d'ús: Resoldre Problema k-NN

Cas d'ús: RESOLDRE PROBLEMA K-NN	
Descripció	Resoldre un problema del tipus k-NN. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització de la xarxa s'ha d'haver calculat. - Ha d'existir almenys un punt de consulta. - Ha d'existir almenys una facilitat. - Cada facilitat ha de tenir assignat un color.
Flux Principal	1. Seleccionar punt de consulta. 2. Introduir tipus de facilitats a buscar. 3. Introduir nombre k de facilitats a buscar. 4. Calcular diagrama Voronoi ordre k sobre les facilitats (veure cas d'ús). 5. Obtenir punt de parametrització del punt de consulta seleccionat. 6. **Per a cada una de les k facilitats que hem de trobar**: 6.1. Obtenir capa k del diagrama de Voronoi. 6.2. Obtenir color de la posició del punt de consulta en la capa k del diagrama de Voronoi. 6.3. Buscar facilitat que té aquest color.
Flux Alternatiu	2. Si no hi ha cap facilitat del tipus indicat mostrar missatge d'avís. 3. El nombre k ha de ser com a mínim 1 i com a màxim el nombre de facilitats del tipus que busquem. Si el nombre k és superior mostrar missatge d'avís i buscar les que es puguin.
Postcondició	Problema k-NN resolt correctament.
Observacions	Si s'indica un tipus concret de facilitat només s'utilitzaran les d'aquells tipus per a fer els càlculs.

3.1.4.18. Cas d'ús: Crear Estructures de Dades

Cas d'ús: CREAR ESTRUCTURES DE DADES	
Descripció	Crear estructures de dades per ordenar els objectes que apareixen a la xarxa de carreteres.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. **Crear QuadTree de vèrtexs**: 1.1. Calcular caixa englobant del QuadTree. 1.2. Crear node arrel. 1.3. **Per a cada vèrtexs de la xarxa de carreteres**: 1.3.1. Afegir vèrtex al node arrel. 2. **Crear RTree d'arestes**: 2.1. Calcular caixa englobant de l'RTree. 2.2. Crear node arrel. 2.3. **Per a cada aresta de la xarxa de carreteres**: 2.3.1. Transformar l'aresta en un rectangle. 2.3.2. Afegir l'aresta al node arrel. 3. **Crear QuadTree d'objectes**: 3.1. Calcular caixa englobant del QuadTree. 3.2. Crear node arrel. 3.3. **Per a cada facilitat de la xarxa de carreteres**: 3.3.1. Afegir facilitat al node arrel. 3.4. **Per a cada punt de consulta de la xarxa de carreteres**: 3.4.1. Afegir punt de consulta al node arrel.
Flux Alternatiu	3. Si no existeix cap facilitat ni punt de consulta, no es crea.
Postcondició	QuadTree de vèrtexs, RTree d'arestes i QuadTree d'objecte creats correctament.
Observacions	- En els punts 1.1 i 3.1 es crearà una caixa englobant. Aquesta caixa ha de tenir una forma quadrada, ha de ser el més petita possible i ha de contenir tots els vèrtexs. - En els punts 1.3.1, 2.3.2, 3.3.1 i 3.4.1. s'afegeixen tots els objectes dins l'arrel. Aquesta ja s'encarrega de crear els seus fills i així recursivament fins crear l'arbre. - En el punt 2.1 es crearà una caixa englobant. Aquesta caixa ha de tenir forma rectangular (no ha de ser quadrada). A més a més, igual que en el cas anterior, ha de ser mínima i contenir totes les arestes.

3.1.4.19. Cas d'ús: Marcar Zona

Cas d'ús: MARCAR ZONA	
Descripció	Seleccionar una zona de la xarxa de carreteres (sub-xarxa) conservant els objectes de la xarxa original. Aquesta funcionalitat ens permetrà resoldre problemes dins d'aquesta zona permetent un càlcul més ràpid.
Actor	Usuari.
Precondició	La xarxa de carreteres està creada.
Flux Principal	1. Marcar un punt a la pantalla. 2. Marcar un segon punt a la pantalla formant un rectangle amb el primer punt. 3. Obtenir els vèrtexs continguts dins el rectangle utilitzant el QuadTree de vèrtexs. 4. **Per a cada vèrtexs de la zona**: 4.1. Fer una còpia del vèrtex. 4.2. Agafar les arestes de dins la zona. 4.3. **Per a cada aresta del nou vèrtex**: 4.3.1. Obtenir objectes que contenen. 5. **Si el nou graf no és connex**: 5.1. Trobar sub-grafs connexos. 5.2. Seleccionar el graf amb més vèrtexs. 5.3. Eliminar els altres grafs. 6. Crear graf de la zona. 7. Crear estructures de dades (veure cas d'ús). 8. Crear parametrització de la xarxa (veure cas d'ús). 9. Assignar el graf de la zona com el graf actual de l'aplicació.
Flux Alternatiu	3. Si no hi ha cap vèrtexs dins el rectangle, l'usuari tindrà la possibilitat de tornar a marcar el rectangle.
Postcondició	Xarxa de carreteres de la zona creada correctament.
Observacions	- En el punt 4.2 s'agafa, per a cada vèrtex de la zona, aquelles arestes que tenen com a origen aquest vèrtex i com a destí un vèrtex que estigui dins de la zona. D'aquesta manera aconseguim tenir un graf dins el rectangle. - Per a resoldre els problemes de proximitat és important que el graf sigui connex (hi ha camí entre qualsevol parella de vèrtexs) ja que les parts no connexes dificulten els càlculs i no aporten cap millora. - En el punt 7 es creen les diferents estructures de dades que necessita l'aplicació per poder treballar d'una manera més eficient amb la xarxa de carreteres.

	- En el punt 8 es crea la parametrització de la xarxa de carreteres per tal de comprimir-la i facilitar-ne els càlculs posteriors.

3.1.4.20. Cas d'ús: Desmarcar Zona

Cas d'ús: DESMARCAR ZONA	
Descripció	Desmarcar una zona ja creada (sub-xarxa) i tornar a visualitzar la xarxa de carreteres original conservant les modificacions realitzades a la zona.
Actor	Usuari.
Precondició	Zona marcada.
Flux Principal	1. Assignar graf original com el graf actual de l'aplicació. 2. Guardar canvis realitzats al graf de la zona al graf original. 3. Eliminar el graf de la zona.
Flux Alternatiu	
Postcondició	Xarxa de carreteres de la zona eliminat.
Observacions	

3.1.4.21. Cas d'ús: Calcular Cercle

Cas d'ús: CALCULAR CERCLE	
Descripció	Calcular un cercle a partir d'una facilitat. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització de la xarxa s'ha d'haver calculat. - Ha d'existir almenys una facilitat.
Flux Principal	1. Escollir facilitat. 2. Introduir distància (radi del cercle). 3. Calcular camp distàncies de la facilitat (veure cas d'ús). 4. Depenent del camp de distàncies normalitzar el radi del cercle entre [0,1]. 5. Calcular tots els píxels de la parametrització que queden dins el radi del cercle. 6. Obtenir les facilitats que queden dins el radi del cercle.
Flux Alternatiu	2. Si la distància introduïda és tan petita que no té sentit calcular el cercle mostrar missatge d'avís.

Postcondició	Cercle calculat correctament.
Observacions	

3.1.4.22. Cas d'ús: Resoldre Problema Reverse k-NN

Cas d'ús: RESOLDRE PROBLEMA REVERSE K-NN	
Descripció	Resoldre un problema del tipus Reverse k-NN. Aquesta funcionalitat es calcula sobre la parametrització de la xarxa.
Actor	Usuari.
Precondició	- El graf de la xarxa s'ha d'haver creat correctament. - La parametrització de la xarxas'ha d'haver calculat. - Han d'existir almenys dues facilitats.
Flux Principal	1. Seleccionar facilitat. 2. Introduir tipus de facilitats a buscar. 3. Introduir valor k. 4. **Per a cada facilitat**: 4.1. Calcular k-NN sobre la facilitat actual utilitzant k+1 (veure cas d'ús). 4.2. Calcular la distància entre la facilitat actual i la facilitat que ens retorna el k-NN. 4.3. Calcular cercle sobre la facilitat actual utilitzant la distància trobada (veure cas d'ús). 4.4. Guardar cercle. 5. Obtenir punt de la parametrització que correspon a la facilitat inicial. 6. **Per a cada facilitat**: 6.1. Si el punt de la facilitat inicial queda dins el cercle de la facilitat actual, la facilitat actual forma part del resultat.
Flux Alternatiu	2. Si no hi ha cap facilitat del tipus a buscar mostrar missatge d'avís. 3. El valor k ha de ser com a mínim 1 i com a màxim el nombre de facilitats del tipus que busquem.
Postcondició	Problema Reverse k-NN resolt correctament.
Observacions	El valor k indica la proximitat de la facilitat que busquem. Per exemple, si k=1 buscarem la facilitat més propera, si k=2 buscarem la segona facilitat més propera, etc.

3.1.5. Diagrama de classes de domini

Els diagrames de classes són els més utilitzats en el modelat de sistemes orientats a objectes. S'utilitzen per tal d'explorar conceptes del domini del problema.

Un diagrama de classes proporciona una visió estàtica del sistema a desenvolupar ja que mostra les classes que interactuen sense mostrar què passa quan interactuen entre elles (Els diagrames que mostren com interactuen les classes entre elles són els diagrames de seqüència i de col·laboració que es troben en el següent capítol).

Tot seguit es mostra un model de classes preliminar (*figura 3.3*) on podem veure els objectes, les classes i les interaccions entre elles.

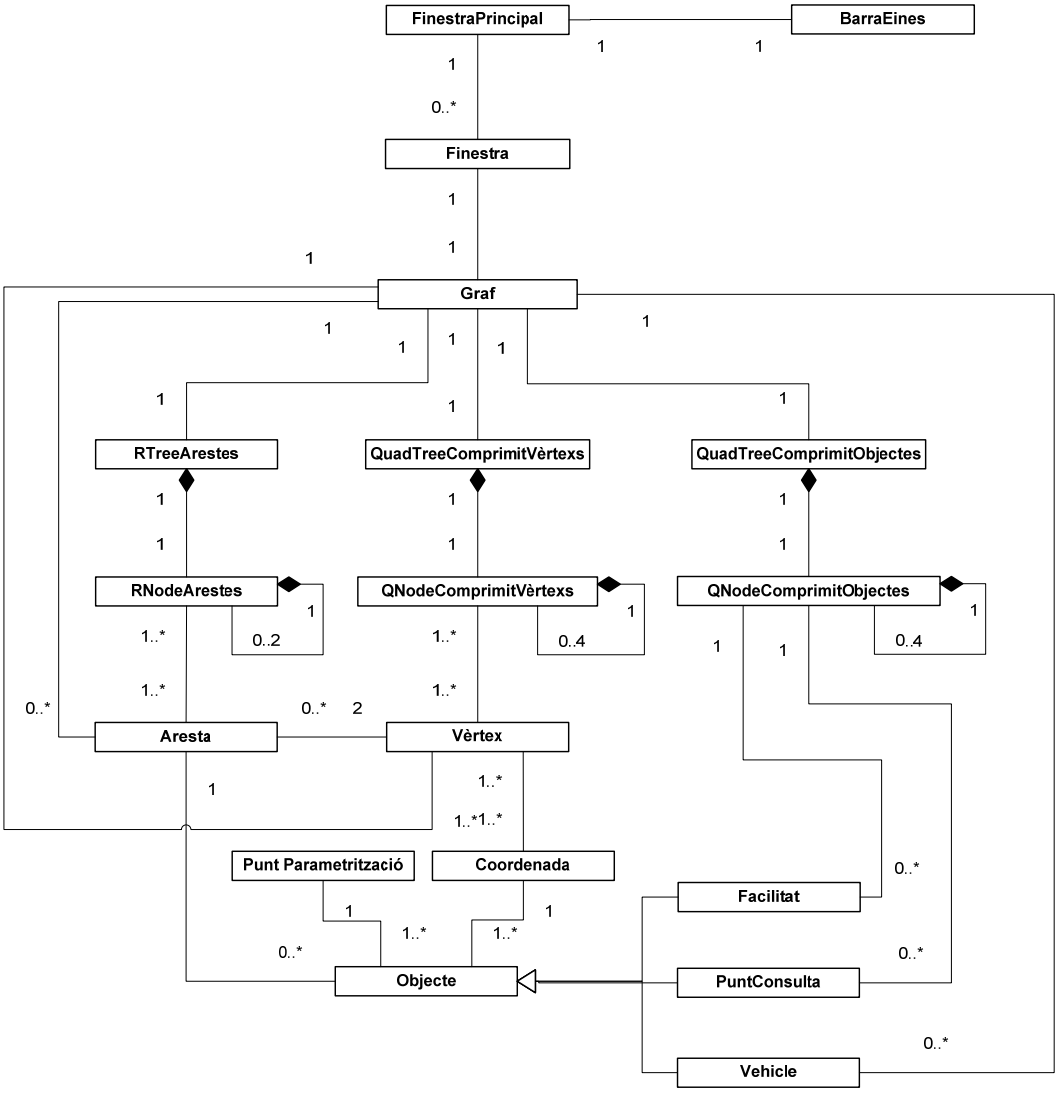


Figura 3.3: Diagrama de classes de domini

3.2. Requeriments no funcionals

Els requeriments no funcionals del sistema són aquells que fan referència a restriccions del tipus: disponibilitat de recursos, seguretat, interfícies externes (hardware i software), entre d'altres. Aquestes condicions ens permetran executar l'aplicació sense cap problema.

L'aplicació cal que sigui executada en un entorn que disposi del sistema operatiu Linux. El motiu pel qual es va escollir implementar-lo sobre aquest software va ser per motius de fiabilitat i coneixements, ja que la majoria de pràctiques relacionades amb la programació durant tota la carrera s'han dut a terme a través d'aquest sistema operatiu.

Un dels requisits que cal complir per poder executar el programa implementat és el fet de disposar de les llibreries gràfiques de les Qt4. Aquesta restricció és deguda a la interfície gràfica que es troba implementada a través d'aquestes.

Per compilar l'aplicació sense problemes es pot fer utilitzant un programa anomenat Qt Designer. Aquest permet crear un "projecte" en el qual cal incloure cadascuna de les classes utilitzades i finalment crear un document makefile que permet compilar l'aplicació. Una altra manera de compilar-lo és utilitzant la comanda qmake.

El temps de resposta del programa depèn lògicament del tipus de memòria de la qual disposi l'ordinador que s'utilitzi. Els problemes de rapidesa de càlcul es fan presents a l'hora de resoldre alguns dels problemes de proximitat depenent del mida de la xarxa de carreteres i del nombre d'objectes que hi hagi. Una altra peça molt important és la targeta gràfica utilitzada, ja que en depenen alguns dels càlculs que fem i la visualització de la xarxa.

4. Anàlisi

L'objectiu de l'anàlisi és obtenir una comprensió precisa de les necessitats del sistema, és a dir, s'encarrega de la investigació del problema a resoldre (*que*), però no s'interessa en trobar-ne una solució (*com*). En aquest procés ens ocupem de traduir els requeriments comentats en el capítol anterior a un llenguatge més formal segons l'Enginyeria del Programari.

El propòsit de l'anàlisi orientat a objectes és definir totes les classes que són rellevants pel problema, així com els atributs i operacions associats a cadascuna de les classes.

L'anàlisi es centra en les vistes d'usuari, com poden ser els diagrames de casos d'ús, i estructurals, com poden ser els diagrames d'interacció que defineixen les classes i les associacions entre elles. Aquests diagrames estan modelats amb UML.

El conjunt format entre els models creats durant les fases de definició de requeriments i anàlisi formaran el domini del problema.

4.1. Diagrames de casos d'ús refinats

A partir del diagrama de casos d'ús general del capítol anterior, podem veure que hi ha casos d'ús que caldria estudiar amb més detall per tal d'entendre amb més exactitud el seu funcionament. A continuació s'exposen amb un nivell més elevat de refinament aquells que es consideren més importants, o bé, més costosos pel que fa als requeriments de que disposen.

4.1.1. Cas d'ús: Carregar Fitxer

Una de les funcionalitats més importants de l'aplicació consisteix en carregar xarxes de carreteres des d'un fitxer que haurà de tenir l'extensió i el format TIGER/Lines.

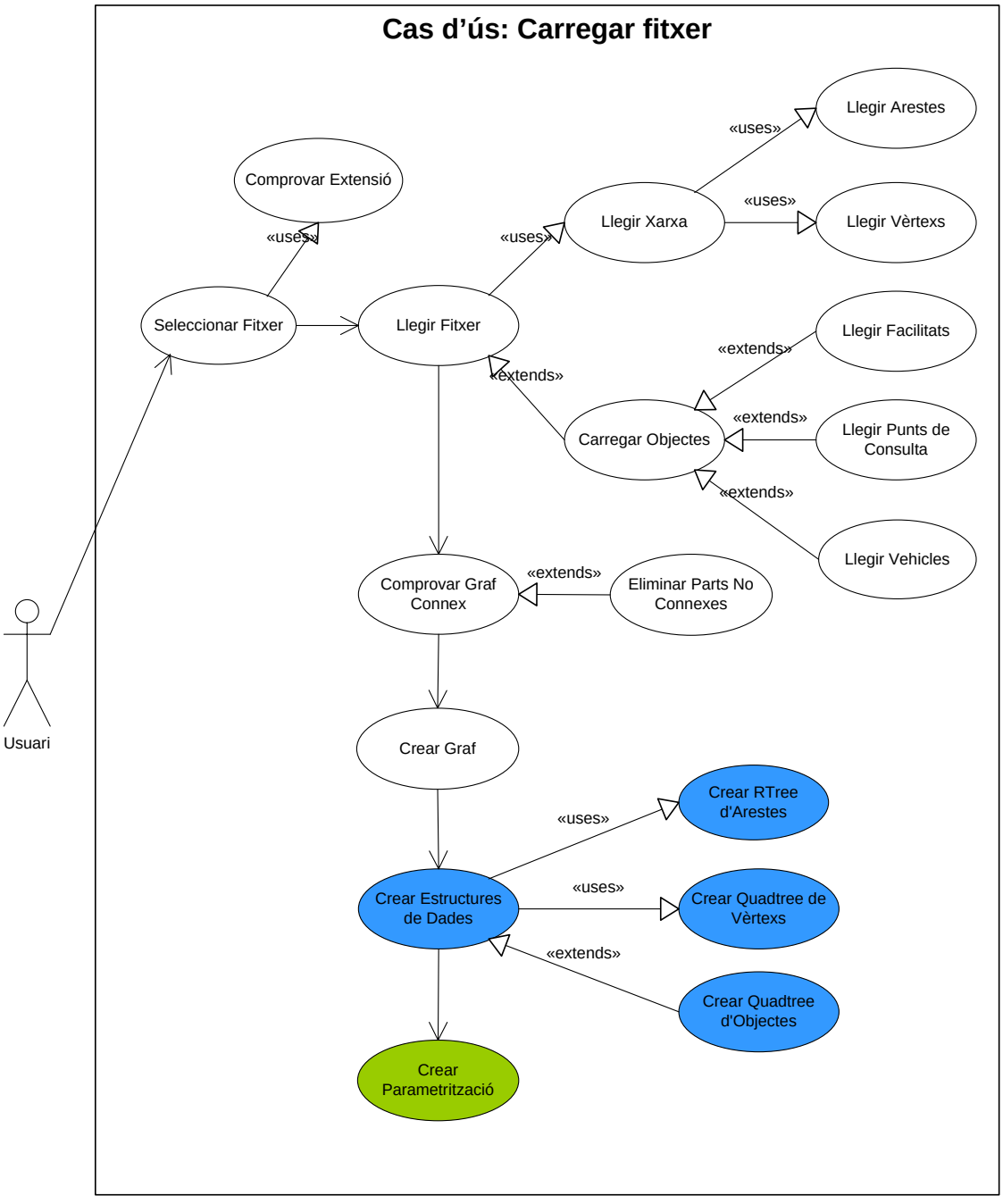


Figura 4.1: Diagrama cas d'ús: Carregar fitxer

El procés a seguir per a carregar una xarxa de carreteres és el següent:

1. El primer que ha de fer l'usuari és seleccionar quin fitxer vol carregar. Un cop l'hagi seleccionat, l'aplicació comprovarà que el fitxer tingui la extensió correcte. En cas que aquesta fos errònia, l'aplicació mostrarà un missatge d'error informant a l'usuari que ha de seleccionar un fitxer del tipus adequat.

2. Un cop es tingui el fitxer de text, s'obrirà i es llegirà línia per línia tots els vèrtexs que contingui. Cal remarcar que el fitxer té un format TIGER/Lines i aquest conté un vèrtex a cada línia. Per a cada vèrtex es llegirà la seva coordenada (longitud i latitud) i es crearà. Tot seguit, es llegiran les arestes adjacents a cada vèrtex i el seu tipus i es crearan.

3. Un cop s'hagin llegit tots els vèrtexs i arestes, es buscarà si existeix un fitxer d'objectes per a poder carregar-los. Aquest fitxer haurà de tenir el mateix nom que el fitxer que conté la xarxa de carreteres però amb una extensió diferent. En cas que existeixi aquest fitxer, s'obrirà i s'anirà llegint línia per línia. Cada línia conté un objecte, ja sigui facilitat, punt de consulta o car.

4. El següent pas consisteix en comprovar que el graf sigui connex, és a dir, que existeixi un camí entre cada parella de vèrtexs. Realitzar aquest pas és molt important ja que les parts no connexes dificulten els càlculs. Un cop localitzats tots els sub-grafs connexos de la xarxa principal, s'escull el més gran i s'eliminen tots els altres. Cal remarcar que realitzant aquesta operació no s'eliminen grans quantitats de vèrtexs, sinó que es neteja petits grups de carrers mal formats que els fitxers TIGER/Lines conté.

5. Un cop es té la llista de vèrtexs i arestes correcte i la possible llista d'objectes, es crea el graf.

6. Un cop s'ha creat el graf, es construeixen vàries estructures de dades per a cadascun dels seus objectes (veure cas d'ús):

   6.1. Es crea un QuadTree de vèrtexs per a ordenar els vèrtexs.

   6.2. Es crea un RTree d'arestes per a ordenar les arestes.

   6.3. Es crea un QuadTree d'objectes per a ordenar els objectes del tipus facilitats i punts de consulta.

7. Es crea la parametrització de la xarxa de carreteres per permetre compactar la xarxa i poder-hi treballar de forma més precisa i còmode (veure cas d'ús).

4.1.2. Cas d'ús: Carregar Objectes

La funcionalitat carregar objectes és una extensió de la funcionalitat carregar fitxer. Aquesta només s'utilitzarà en cas que, després de carregar una xarxa de carreteres, existeixi un altre fitxer que guardi informació sobre els objectes a afegir.

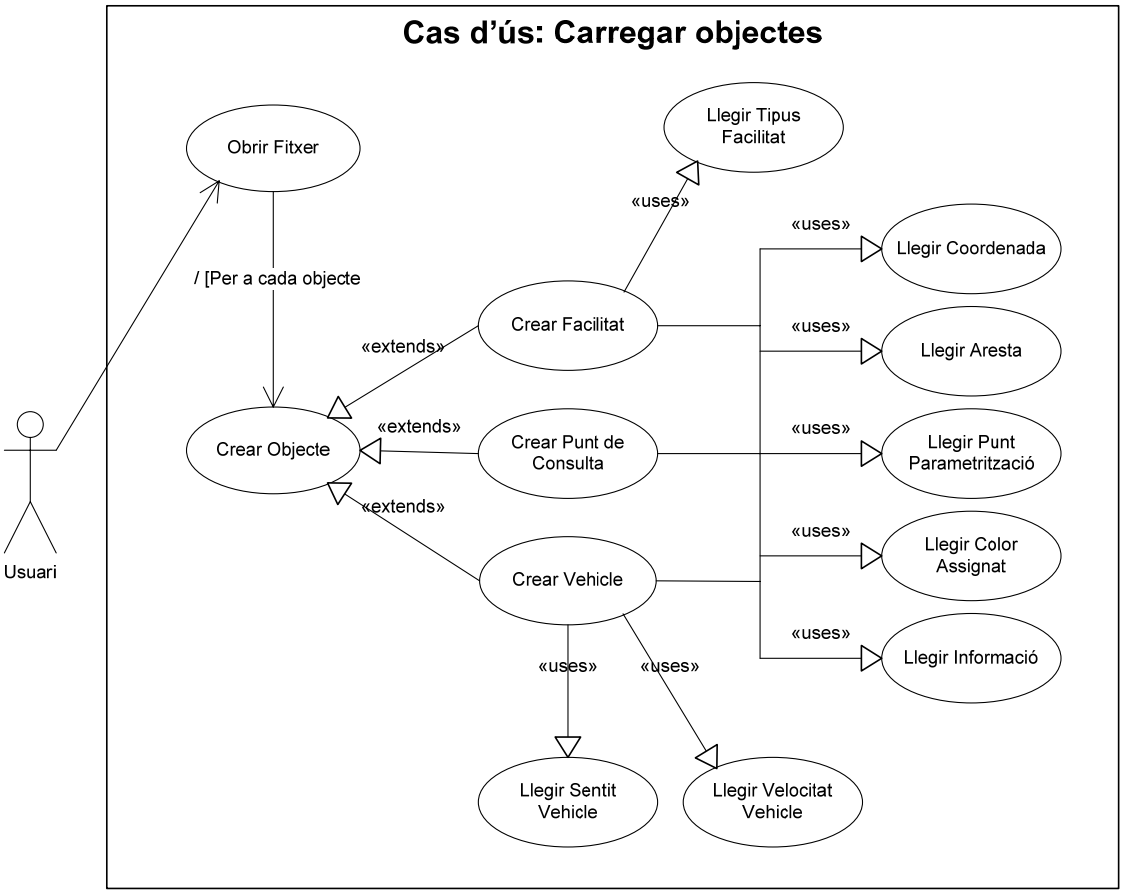


Figura 4.2: Diagrama cas d'ús: Carregar objectes

El procés a seguir per a carregar objectes és el següent:

1. Primerament s'obre el fitxer d'objectes on, a cada una de les línies, hi ha la informació necessària per a crear un objecte.

2. Així doncs, es va llegint el fitxer línia per línia creant els objectes corresponents. Per a ajudar a identificar el tipus d'objecte que s'està llegint, el primer caràcter de cada línia del fitxer informa si és una facilitat, un punt de consulta o un car.

   2.1. En el cas que el tipus d'objecte sigui una facilitat, es llegirà la seva coordenada, l'aresta que forma part, el punt de parametrització, el color que té assignada, el nom, la adreça, la web i, finalment, el tipus de facilitat (gasolinera, hotel, central nuclear, etc.).

2.2. En el cas que el tipus sigui un punt de consulta, es llegirà la seva coordenada, l'aresta que forma part, el punt de parametrització, el color que té assignat i el seu nom.

2.3. Finalment, si el tipus és un car, es llegirà la seva coordenada, l'aresta que forma part, el sentit cap on es dirigeix el car, el punt de parametrització, el color que té assignat i la seva velocitat.

4.1.3. Cas d'ús: Afegir Objectes

La funcionalitat afegir objectes, com el seu propi nom indica, és la que permet a l'usuari afegir facilitats, punts de consulta o vehicles a la xarxa de carreteres. Per a fer-ho tindrà la possibilitat d'afegir-les una a una introduint les dades exactes o bé escollir la opció de crear-ne vàries d'aleatòries.

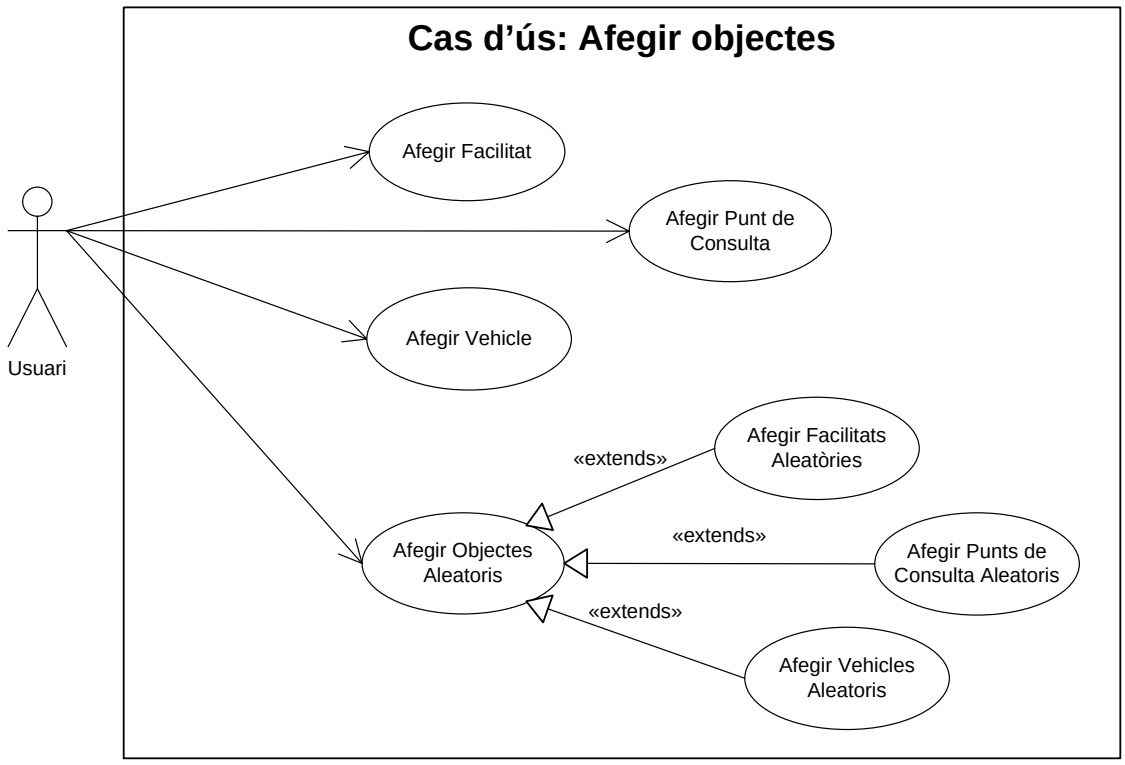


Figura 4.3: Diagrama cas d'ús: Afegir objectes

Els diferents processos a seguir per a crear objectes són els següents:

1. Afegir un objecte introduint les seves dades. L'usuari tindrà la opció d'escollir quin tipus d'objecte vol crear (facilitat, punt de consulta o car) i escollir la seva coordenada exacta dins la xarxa. Tot seguit tindrà la possibilitat d'introduir cada una de les seves dades. A la *figura 4.3* podem observar que les funcionalitats que permeten afegir-les d'aquesta forma són 'Afegir Facilitat', 'Afegir Punt de consulta' i 'Afegir Vehicle'. Degut que el

refinament d'aquests tres casos d'ús són molt similars, s'ha optat per a refinar només 'Afegir Facilitat'.

2. L'altra opció que té l'usuari és afegir varis objectes aleatòriament. Per a fer-ho haurà d'indicar quants objectes vol introduir de cada tipus. A més a més, per a les facilitats podrà elegir quin tipus vol que tinguin i per els vehicles quina vol que sigui la seva velocitat. Igual que en el cas anterior, els casos d'ús 'Afegir Facilitats Aleatòries', 'Afegir Punts de consulta Aleatoris' i 'Afegir Vehicles Aleatoris' són molt semblants i s'ha optat per a refinar només el primer.

4.1.4. Cas d'ús: Afegir Facilitat

Com hem explicat a l'apartat anterior, aquesta permet a l'usuari crear una facilitat introduint totes les seves dades.

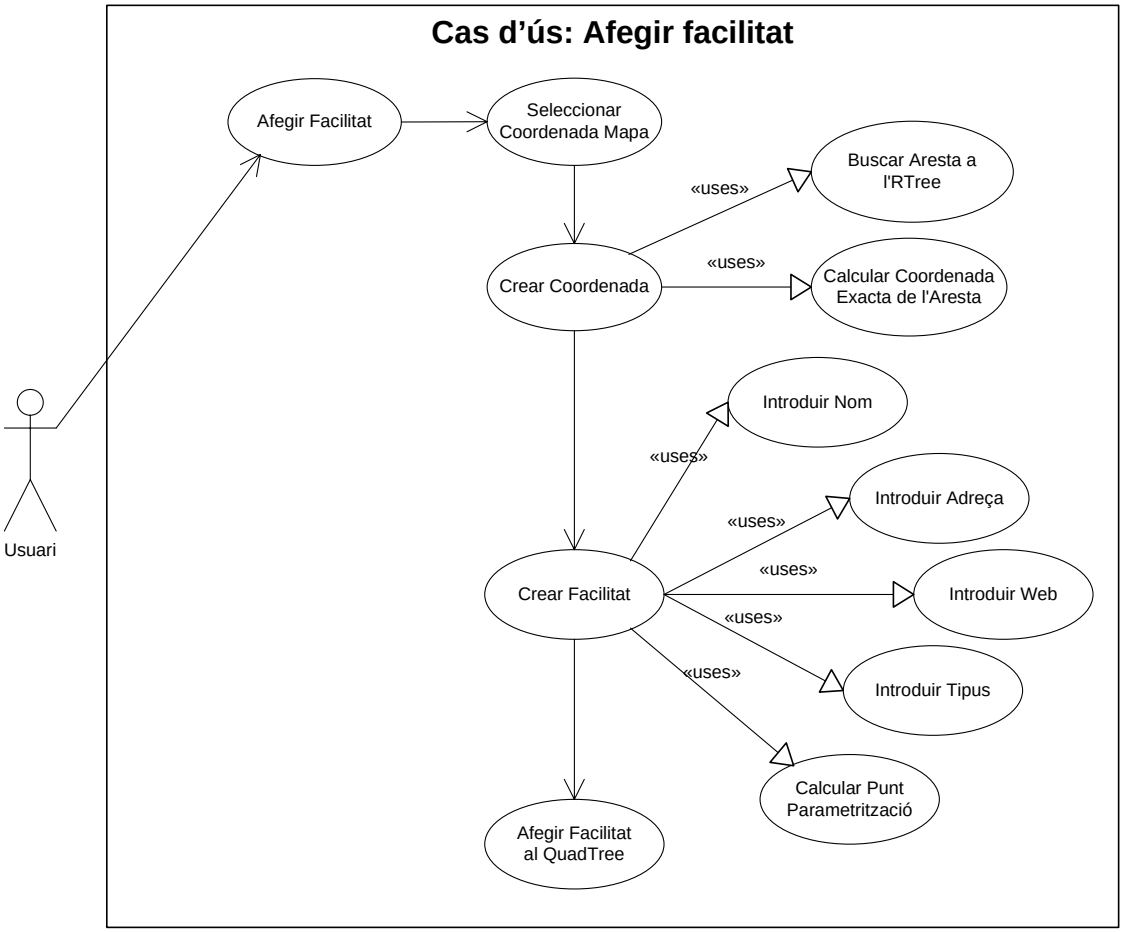


Figura 4.4: Diagrama cas d'ús: Afegir facilitat

A continuació podem veure els diferents processos a seguir:

1. Inicialment, l'usuari ha de marcar una punt sobre la xarxa de carreteres on vulgui afegir la facilitat.

2. A partir d'aquest punt, s'haurà de buscar l'aresta que estigui tocant a aquest punt (o que estigui molt a prop). Per a fer-ho, utilitzarem la estructura de dades RTree creada prèviament amb la funcionalitat 'Carregar fitxer'. Un cop es tingui l'aresta es buscarà la coordenada exacta sobre aquest que estigui el més a prop possible del punt introduït per l'usuari. Per a trobar aquest punt s'utilitzarà un algorisme matemàtic.

3. Tot seguit se li demanarà a l'usuari que introdueixi el nom, adreça i web de la facilitat. També se li demanarà que li assigni un tipus (gasolinera, hotel, restaurant, etc.). Un cop l'usuari hagi informat totes les dades es calcularà quin és el punt de parametrització de la facilitat.

4. Finalment, un cop s'hagi creat la facilitat amb totes les dades, s'haurà d'introduir dins el QuadTree d'objectes.

4.1.5. Cas d'ús: Afegir Facilitats Aleatòries

Com hem explicat en apartats anteriors, aquest cas d'ús permet a l'usuari crear vàries facilitats aleatòriament sobre la xarxa de carreteres.

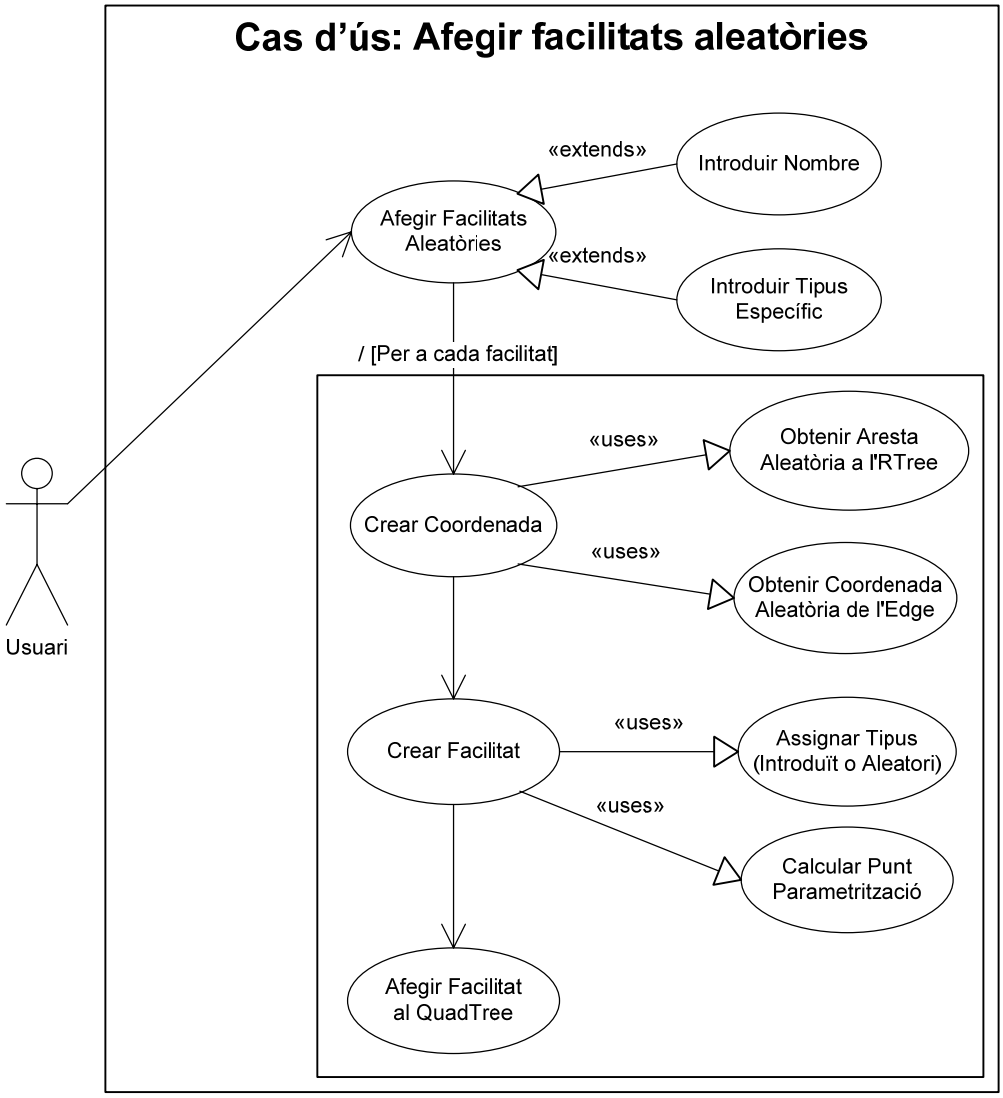


Figura 4.5: Diagrama cas d'ús: Afegir facilitats aleatòries

Els diferents processos a seguir són els següents:

1. El primer que haurà de fer l'usuari en aquest cas és introduir el nombre de facilitats que vol afegir a la xarxa de carreteres. Tot seguit, tindrà la possibilitat d'escollir el tipus de facilitats que voldrà crear (totes les facilitats amb el tipus escollit per l'usuari o totes les facilitats amb tipus aleatoris).

2. Per a cada facilitat que es vulgui introduir a la xarxa de carreteres es crea una coordenada aleatòria. Per a trobar-la, es selecciona una aresta aleatori mitjançant l'RTree i es busca una coordenada (també aleatòria) sobre aquesta aresta.

3. Es crea la facilitat introduint-li la coordenada calculada al punt anterior i se li assigna el tipus. Aquest tipus, com ja hem explicat anteriorment, pot ser el que ha escollit l'usuari al punt 1 o l'haurem de calcular aleatòriament. Per acabar de crear la facilitat, es calcularà el seu punt de parametrització.

4. Finalment, s'introduirà la facilitat dins el QuadTree d'objectes.

4.1.6. Cas d'ús: Calcular parametrització de la xarxa

Al calcular la parametrització de la xarxa el que es fa és mapejar cadascuna de les arestes en una reixa rectangular per posteriorment poder-hi realitzar els càlculs d'una forma més eficient i precisa.

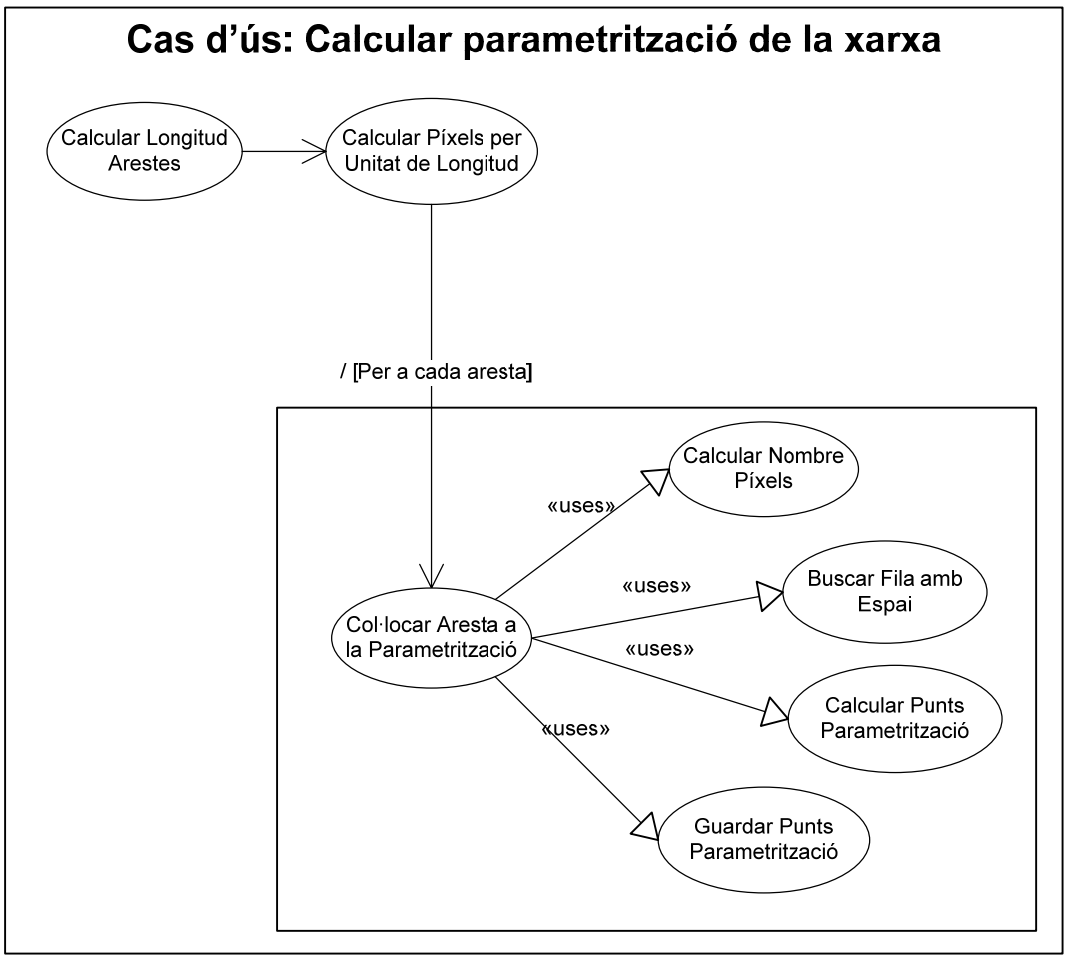


Figura 4.6: Diagrama cas d'ús: Calcular parametrització de la xarxa

Per tal de calcular la parametrització de la xarxa el procés que cal seguir és el següent:

1. Calcular la longitud total de totes les arestes del graf.

2. A partir de la longitud total i dels píxels disponibles, calcular els píxels que s'assignaran per unitat de longitud.

3. Tot seguit, per a cada aresta de la xarxa.

    3.1. Mapejar l'aresta a la parametrització. Per a fer-ho cal calcular primer el nombre de píxels que tindrà l'aresta a la parametrització segons la seva mida a la xarxa. Després cal buscar una fila amb prou píxels disponibles i calcular els punts de parametrització inicial i final per a l'aresta. Un cop calculats, es guardaran a l'aresta.

4.1.7. Cas d'ús: Calcular Camí Mínim

Per tal de calcular el camí mínim entre dos punts de la xarxa utilitzarem l'algorisme de camins mínims de Dijkstra. Per a fer-ho utilitzarem com a pesos les distàncies que hi ha entre els vèrtexs.

Per a resoldre els problemes de proximitat plantejats necessitarem calcular el camí mínim des d'una facilitat fins a tots els vèrtexs del graf. A més a més, incorporarem una funcionalitat per tal que l'usuari pugui calcular el camí més curt des de la seva posició fins a una facilitat del mapa, és a dir, des d'un punt de consulta fins a una facilitat. En aquest cas, el càlcul que es farà serà el mateix però a partir del punt de consulta.

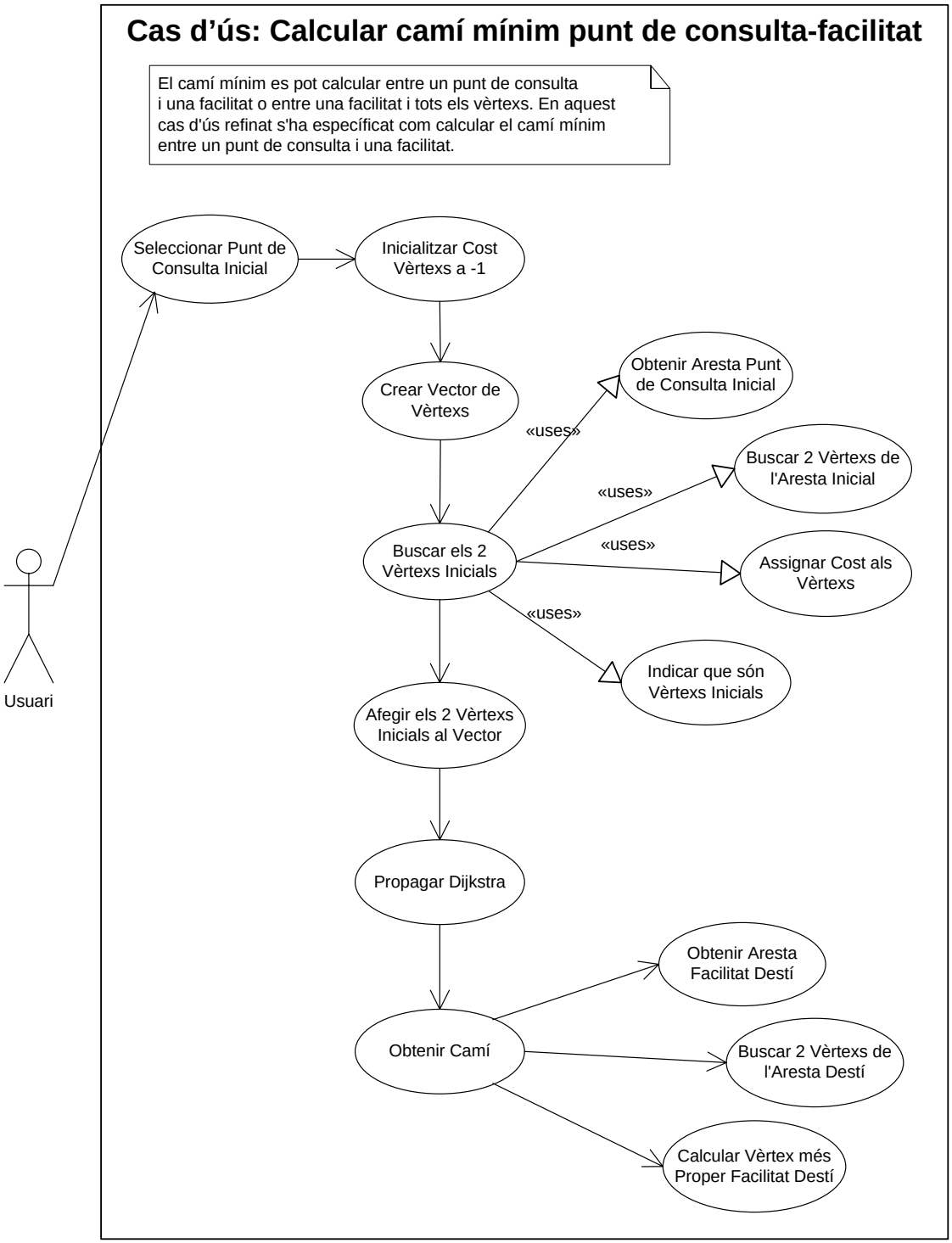


Figura 4.7: Diagrama cas d'ús: Calcular camí mínim

Per tal de calcular el camí mínim entre un punt de consulta i una facilitat el procés que cal seguir és el següent:

1. L'usuari ha de seleccionar el punt de consulta inicial.

2. Inicialitzar el cost de tots els vèrtexs a -1.

3. Tot seguit cal crear un vector de vèrtexs on es posaran tots els vèrtexs que cal tractar.

4. Per començar a calcular el camí mínim es necessita un vèrtex inicial. Com que el punt de consulta està col·locat sobre una aresta, el que es fa es obtenir els dos vèrtexs que defineixen aquesta aresta. A aquestes dues arestes com a cost se'ls assigna la distància que hi ha entre el punt de consulta i el vèrtex i s'indica que són vèrtex inicials i que no tenen cap antecessor.

5. S'afegeixen els dos vèrtexs al vector.

6. Seguidament, s'aplica l'algorisme de Dijkstra per tal de trobar el camí mínim fins a tots els vèrtexs del graf. Per començar a propagar s'utilitzarà un dels dos vèrtexs inicials, exactament el que tingui un cost menor.

7. Finalment el que volem és obtenir el camí mínim del punt de consulta fins a la facilitat. A partir de l'aresta on està col·locada la facilitat mirarem quin dels dos vèrtexs de l'aresta està més proper a la facilitat. A partir d'aquest vèrtex ja podem obtenir el camí fent un procés recursiu d'obtenir l'antecessor del vèrtex fins a arribar al vèrtex marcat com a inicial, i des d'aquest fins a la punt de consulta.

4.1.8. Cas d'ús: Resoldre Problema Aggregate k-NN

Al resoldre un problema del tipus Aggregate k-NN s'obtenen les k facilitats millors situades d'acord amb una funció de distància respecte dels punts de consulta de la xarxa. Definirem 4 funcions diferents:

- **MinMax**: troba les k facilitats que fan mínima la màxima distància.
- **MaxMin**: troba les k facilitats que fan màxima la mínima distància.
- **MinSum**: troba les k facilitats que fan mínima la suma de distàncies.
- **MaxSum**: troba les k facilitats que fan màxima la suma de distàncies.

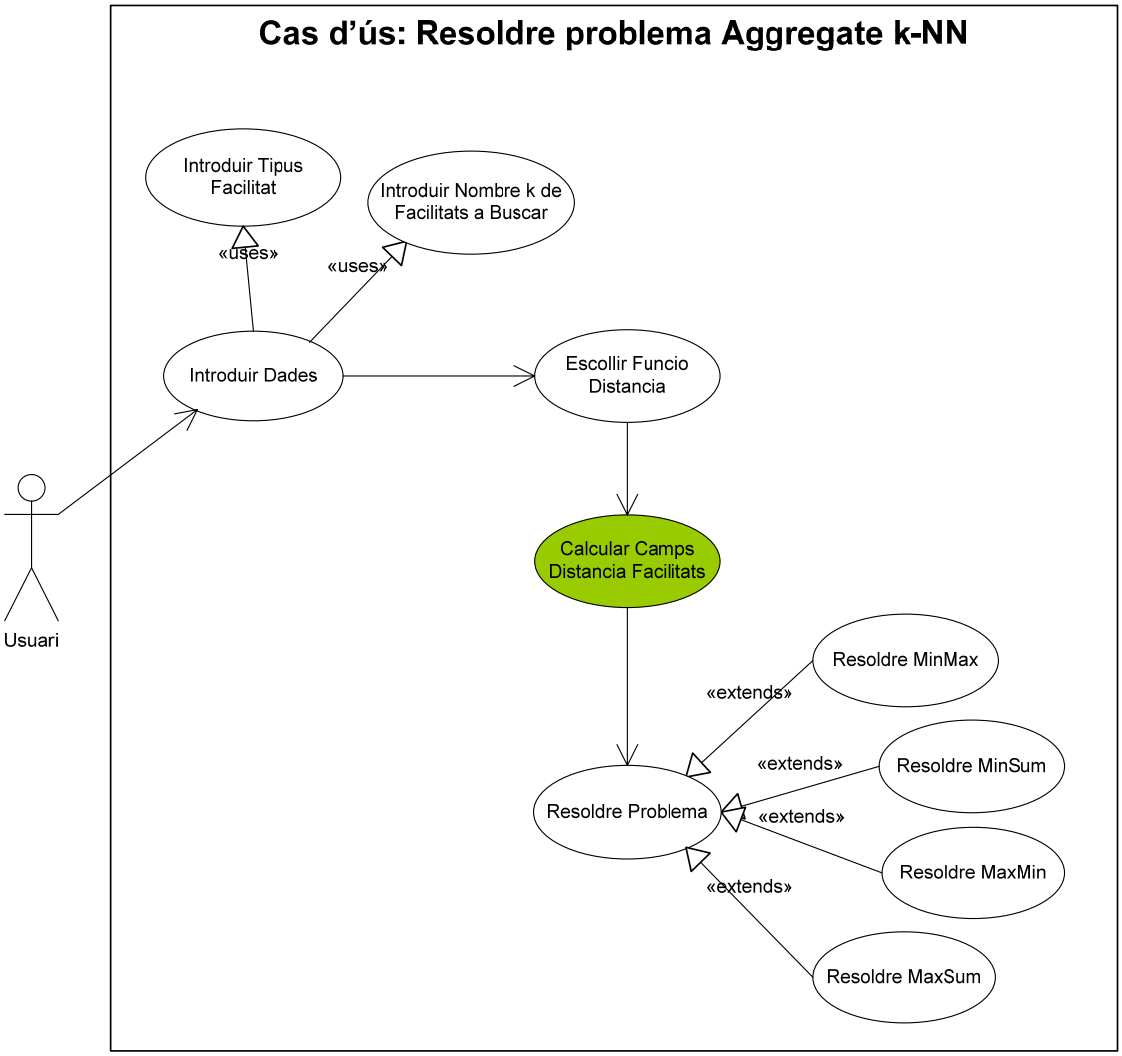


Figura 4.8: Diagrama cas d'ús: Resoldre problema Aggregate k-NN

Per tal de resoldre el problema Aggregate k-NN el procés que cal seguir és el següent:

1. L'usuari ha d'introduir el tipus de facilitats a utilitzar per a resoldre el problema (es pot resoldre amb totes les facilitats o només amb les d'un tipus concret: restaurants, hospitals, etc.) i el nombre k de facilitats a buscar.

2. L'usuari ha d'escollir una funció per a fer el càlcul: MinMax, MaxMin, MinSum o MaxSum.

3. Seguidament, cal calcular el camp de distàncies de cada facilitat.

4. La resolució del problema dependrà de la funció escollida:

4.1. MinMax. Per a cada facilitat es calcula la distància a tots els punts de consulta i es guarda la major. Posteriorment, es seleccionen les k facilitats que tenen una menor distància guardada.

4.2. MaxMin. Per a cada facilitat es calcula la distància a tots els punts de consulta i es guarda la menor. Posteriorment, es seleccionen les k facilitats que tenen una major distància guardada.

4.3. MinSum. Per a cada facilitat es calcula la distància a tots els punts de consulta i es guarda la suma d'aquestes. Posteriorment, es seleccionen les k facilitats que tenen una suma menor guardada.

4.4. MaxSum. Per a cada facilitat es calcula la distància a tots els punts de consulta i es guarda la suma d'aquestes. Posteriorment, es seleccionen les k facilitats que tenen una suma major guardada.

4.1.9. Cas d'ús: Resoldre Problema k-NN

Al resoldre un problema del tipus k-NN s'obtenen les k facilitats més properes a un determinat punt de consulta.

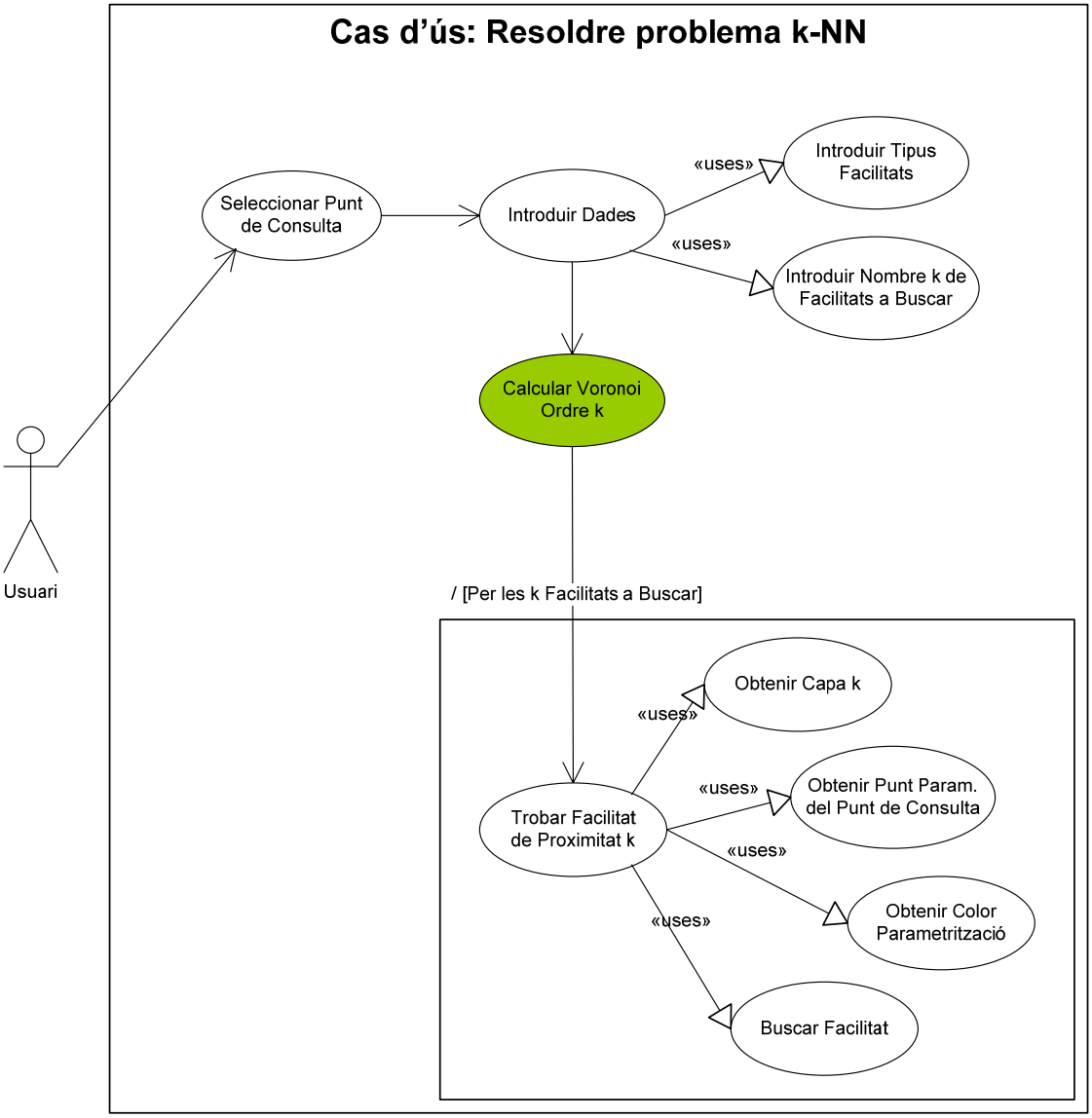


Figura 4.9: Diagrama cas d'ús: Resoldre problema k-NN

Per tal de resoldre el problema k-NN el procés que cal seguir és el següent:

1. L'usuari ha de seleccionar el punt de consulta a partir del qual vol resoldre el problema.

2. L'usuari ha d'introduir el tipus de facilitats a utilitzar per a resoldre el problema (es pot resoldre amb totes les facilitats o només amb les d'un tipus concret: restaurants, hospitals, etc.) i el nombre k de facilitats a buscar.

3. Tot seguit, es calcula el diagrama de Voronoi d'ordre k sobre el tipus de facilitats escollides. (Veure cas d'ús).

4. Finalment, per a cada una de la k facilitats que cal trobar:
   4.1. S'obté la capa k calculada en el diagrama de Voronoi.
   4.2. S'obté el punt de parametrització del punt de consulta inicial.
   4.3. Aquest punt permet buscar sobre la capa k el color amb què s'ha pintat.
   4.4. Es busca la facilitat que té aquest color associat. Aquesta facilitat formarà part del resultat.

4.1.10. Cas d'ús: Crear Estructures De Dades

Aquesta funcionalitat és una de les més importants de l'aplicació ja que és l'encarregada d'ordenar tots els objectes que tenim a la xarxa de carreteres (vèrtexs, arestes i objectes). Com s'ha vist aquestes es creen quan carreguem un fitxer i ja romanen creades fins al final de l'execució.

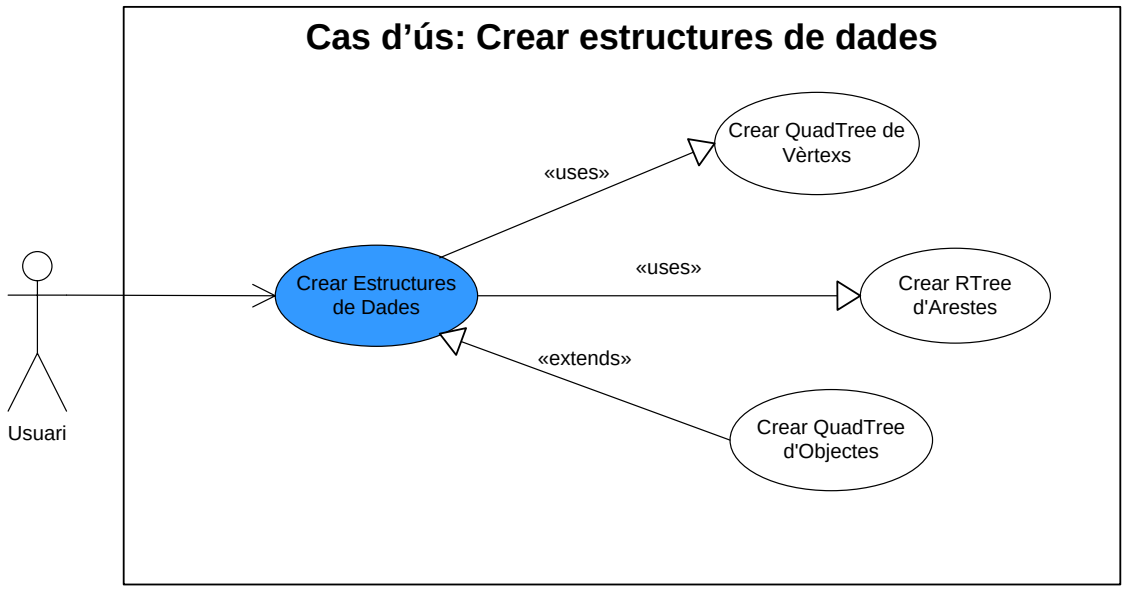


Figura 4.10: Diagrama cas d'ús: Crear estructures de dades

Els diferents processos a seguir per a crear objectes són els següents:

1. Com es pot veure a la figura anterior, aquesta funcionalitat es divideix en tres diferenciades parts. Es pot observar que s'ha de crear un QuadTree de vèrtexs i un RTree d'arestes, mentre que crear un QuadTree d'objectes és opcional (depèn de si existeix el fitxer d'objectes quan es carregui la xarxa de carreteres).

Com que crear cadascuna d'aquestes tres estructures de dades és bastant complex, s'ha optat per refinar només una d'elles per fer-nos una idea del funcionament d'aquestes estructures.

4.1.11. Cas d'ús: Crear QuadTree de Vèrtexs

Crear un QuadTree de vèrtexs és una funcionalitat que s'utilitza quan es creen les estructures de dades. Aquesta consisteix en tenir ordenats dins l'espai 2D tots els vèrtexs que componen la xarxa de carreteres. Aquesta estructura ens servirà per a poder accedir a ells d'una manera ràpida i eficient.

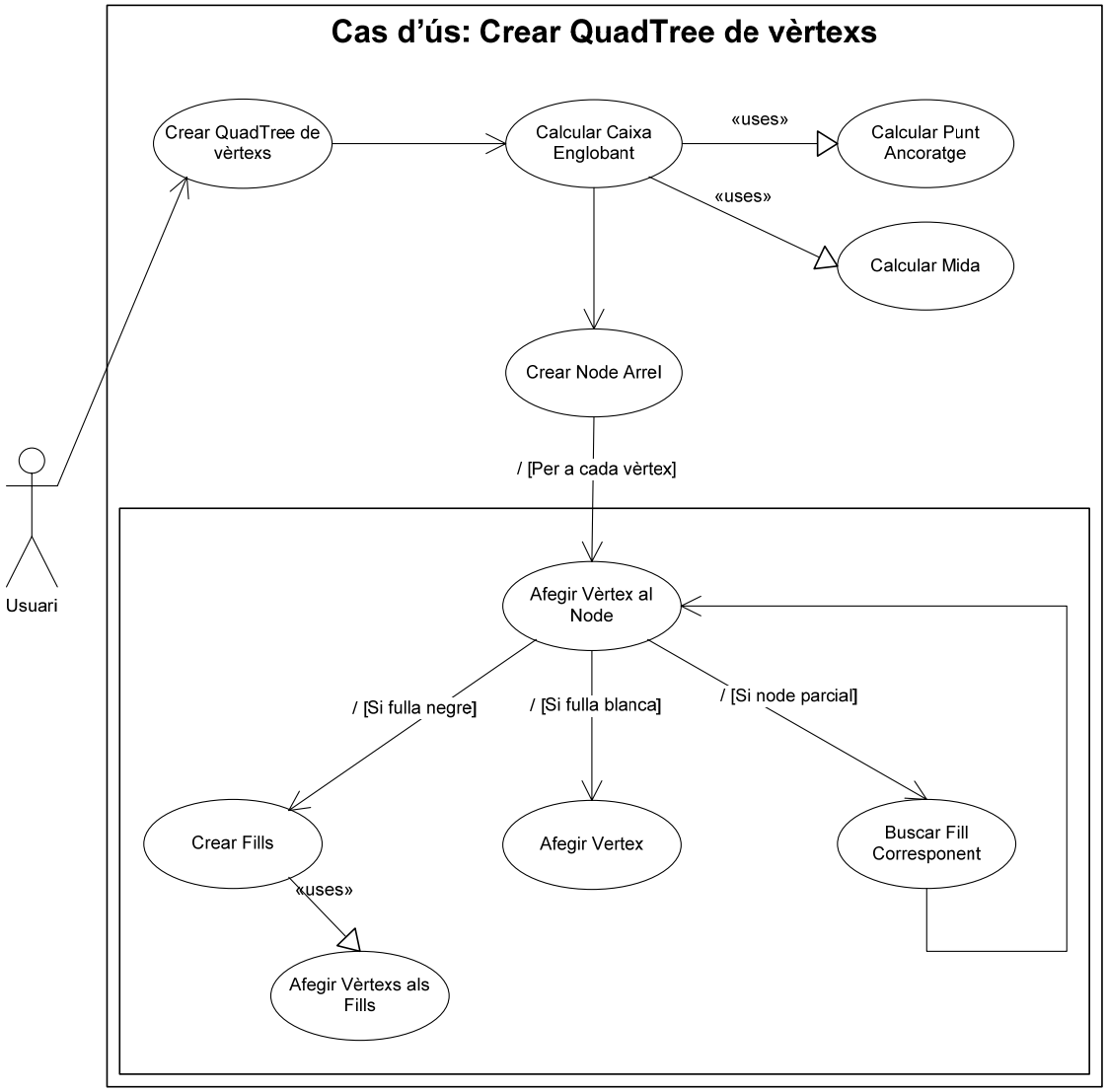


Figura 4.11: Diagrama cas d'ús: Crear QuadTree de vèrtexs

Els processos a seguir per a crear el QuadTree són els següents:

1. Es calcula una caixa englobant mínima, de forma quadrada, suficientment gran per a poder-hi col·locar tots els vèrtexs dins. Per a fer-ho, ens caldrà trobar el punt d'ancoratge (cantonada superior-esquerra) i la seva mida.

2. Crear el node arrel del QuadTree amb el punt d'ancoratge i la mida calculats al punt anterior.

3. Per a cadascun dels vèrtexs, s'aniran introduint dins del node arrel i aquest s'encarregarà d'anar creant l'estructura en forma d'arbre. Quan s'introdueix un vèrtex dins un node es pot trobar amb tres situacions diferents:

    3.1. Si s'afegeix un vèrtex a una fulla negre significa que ja hi ha un element dins i, per tant, s'hauran de crear quatre fills nous (subdividint la zona del node en quatre parts). Un cop es tenen els fills creats, s'afegeixen els dos vèrtexs (el nou que afegim i el que ja hi havia a la fulla) als fills corresponents.

    3.2. Si s'afegeix un vèrtexs a una fulla blanca significa que hi ha una posició lliure i, per tant, s'afegeix sense cap problema.

    3.3. Si s'afegeix un vèrtex dins un node parcial significa que aquest té fills. Per tant, el que s'haurà de fer és veure a quin fill correspon (depenent de la seva coordenada) i anar-lo inserint recursivament fins arribar a algun node fulla.

4.1.12. Cas d'ús: Marcar Zona

Aquesta funcionalitat permet seleccionar una zona de la xarxa de carreteres (de forma rectangular) i creant una sub-xarxa que ens permetrà resoldre problemes només dins aquesta zona fent que els càlculs siguin més ràpids.

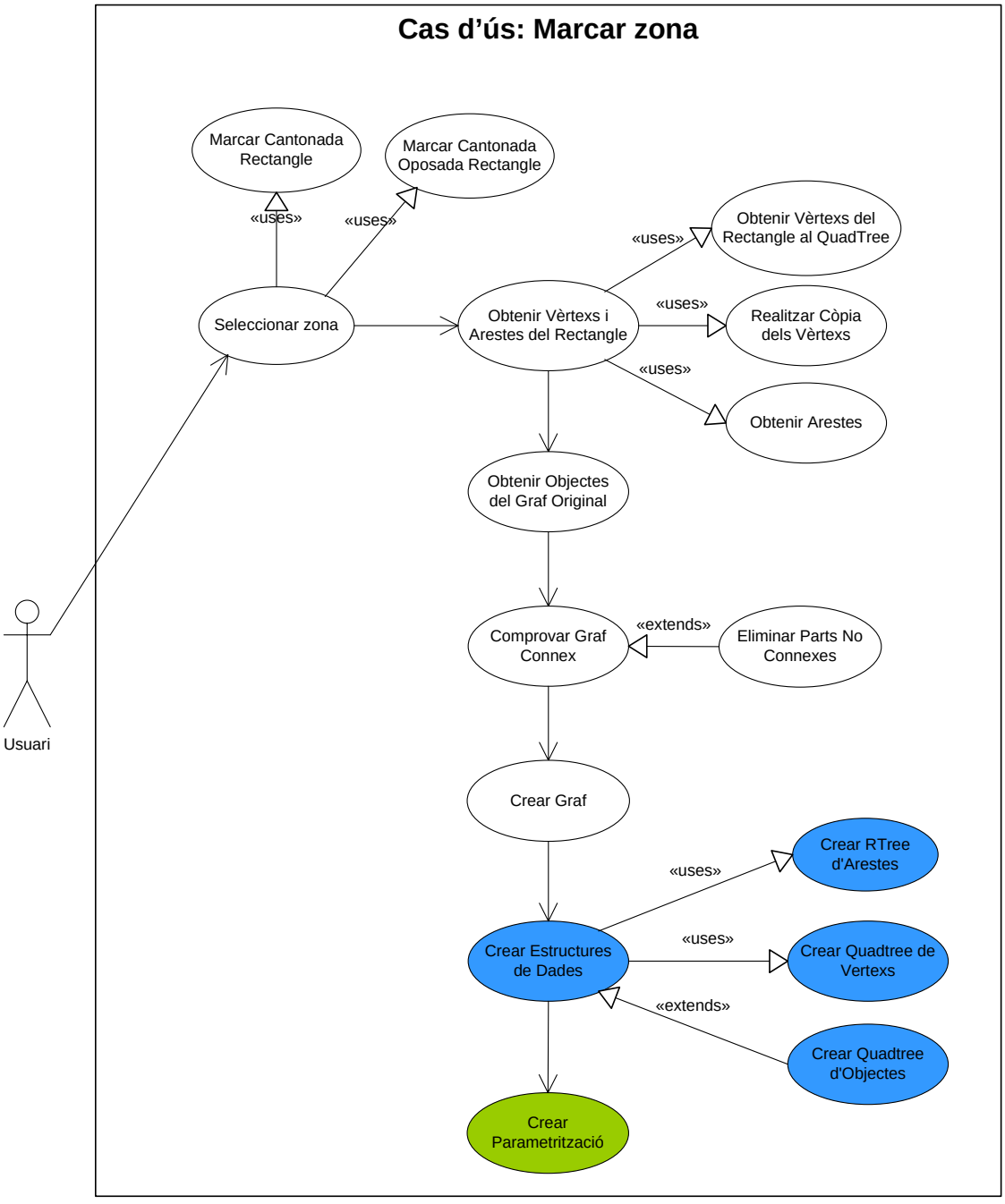


Figura 4.12: Diagrama cas d'ús: Marcar zona

Els processos que cal seguir per a crear una zona són els següents:

1. L'usuari ha de seleccionar la zona. Per fer-ho ha de marcar un rectangle damunt la xarxa de carreteres seleccionant dos punts oposats d'aquest.

2. Mitjançant l'estructura de dades del QuadTree de vèrtexs, s'obtindran tots els vèrtexs que formen part del rectangle marcat per l'usuari. Tot seguit, es realitzarà una còpia d'aquests i se'ls hi assignaran les arestes que també estiguin a la zona (si alguna aresta té algun dels dos vèrtexs que la formen fora del rectangle, no s'utilitzarà).

3. Del conjunt d'arestes que tenim, hem de d'obtenir tots els objectes que formen part d'ells, ja que les necessitarem per a introduir-les al graf de la zona.

4. El següent pas consisteix en comprovar que el graf sigui connex, és a dir, que existeixi un camí entre cada parella de vèrtexs. Realitzar aquest pas és molt important ja que les parts no connexes dificulten els càlculs. Un cop localitzats tots els sub-grafs connexos de la xarxa principal, s'escull el més gran i s'eliminen tots els altres.

5. Un cop es té la llista de vèrtexs i arestes correcte i la possible llista de d'objectes, es crea el graf.

6. Un cop s'ha creat el graf, es construeixen les estructures de dades que hem vist a l'apartat anterior (veure cas d'ús):

7. Es crea la parametrització de la xarxa de carreteres per permetre compactar la xarxa i poder-hi treballar de forma més precisa i còmode (veure cas d'ús).

4.1.13. Cas d'ús: Calcular Cercle

Al calcular un cercle amb un determinat radi al voltant d'una facilitat obtenim totes les facilitats que es troben a distància inferior o igual al radi introduït.

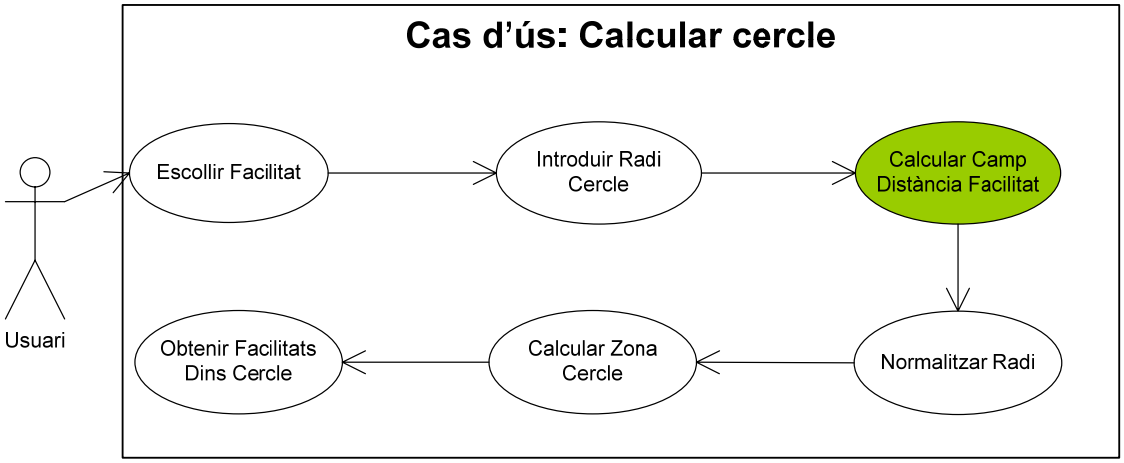


Figura 4.13: Diagrama cas d'ús: Calcular cercle

Per tal de calcular el cercle el procés que cal seguir és el següent:

1. L'usuari ha de seleccionar la facilitat a partir de la qual vol resoldre el problema.

2. L'usuari ha d'introduir el radi del cercle.

3. Tot seguit, es calcula el camp de distàncies a partir de la facilitat inicial (Veure cas d'ús).

4. El radi del cercle l'usuari l'entra en metres i caldrà normalitzar aquesta distància entre [0,1].

5. Al calcular la zona del cercle el que cal fer és a partir del camp distàncies calcular quin són aquells píxels que tenen una distància inferior o igual a la del cercle.

6. Finalment, cal obtenir les facilitats que queden dins el radi del cercle.

4.1.14. Cas d'ús: Resoldre Problema Reverse k-NN

Al resoldre un problema del tipus Reverse k-NN s'obtenen les k facilitats que es trobem més a prop d'una determinada facilitat que de les altres.

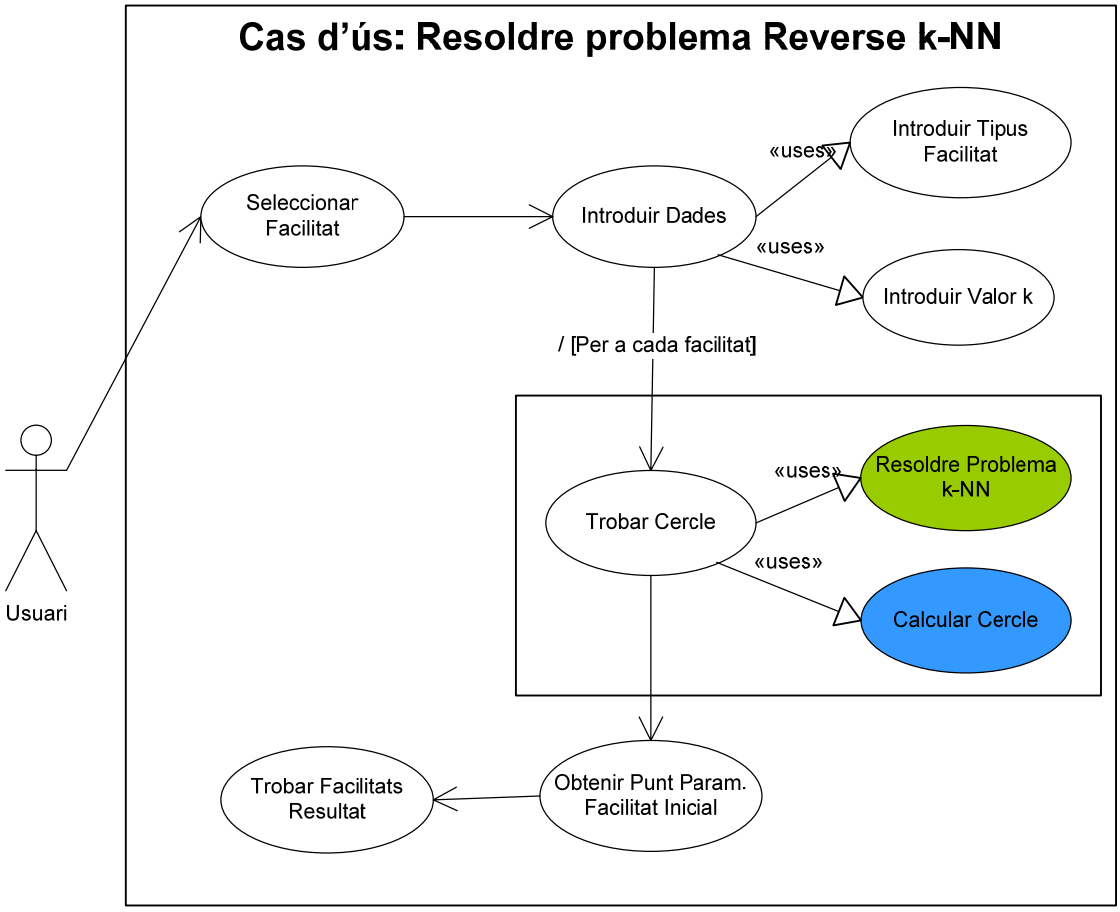


Figura 4.14: Diagrama cas d'ús: Resoldre problema Reverse k-NN

Per tal de resoldre el problema Reverse k-NN el procés que cal seguir és el següent:

1. L'usuari ha de seleccionar la facilitat a partir del qual vol resoldre el problema.

2. L'usuari ha d'introduir el tipus de facilitats a utilitzar per a resoldre el problema (es pot resoldre amb totes les facilitats o només amb les d'un tipus concret: restaurants, hospitals, etc.) i el valor k.

3. Per a cada facilitat cal calcular un cercle.
   3.1. El radi d'aquest cercle és la distància entre la facilitat i la k facilitat més propera a aquesta. Per tal de trobar aquesta facilitat s'utilitza el k-NN (Veure cas d'ús). Es calcula la distància entre la facilitat actual i la facilitat trobada que serà el radi del cercle.

3.2. Es calcula el cercle sobre la facilitat actual amb el radi trobat (Veure cas d'ús). Es guarda aquest cercle.

4. Seguidament, cal obtenir el punt de parametrització que correspon a la facilitat inicial.

5. Finalment, es busquen les facilitats resultants. Per a cada facilitat, es mira si el punt de parametrització de la facilitat inicial queda dins el cercle calculat. Si és així la facilitat amb la que s'ha calculat el cercle formarà part del resultat.

4.2. Diagrama de classes

A partir del diagrama de classes de domini creat al capítol anterior, hem de definir els atributs de cada classe i els seus mètodes.

El diagrama de classes amb els atributs i mètodes és molt gran i és impossible visualitzar-lo complert, així doncs s'exposarà el diagrama de classes sense els atributs i mètodes i tot seguit es mostrarà cada classe per separat on podrem visualitzar-ho amb claredat. A més a més, es comentarà cada classe destacant les principals característiques per tal de fer-les més entenedores.

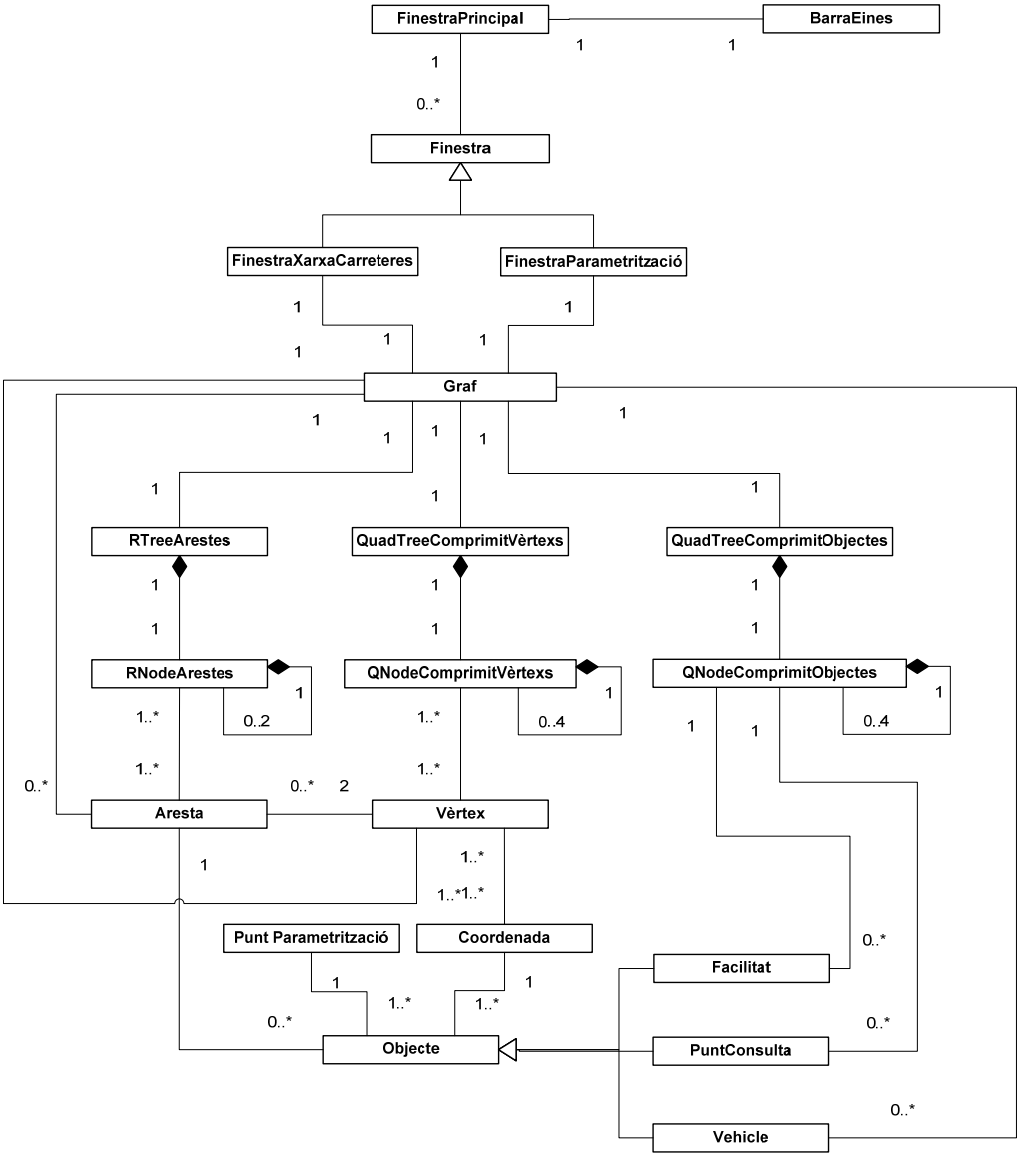


Figura 4.15: Diagrama de classes

Aquest diagrama de classes ha estat millorat respecte el diagrama de classes del domini creat a l'apartat anterior. Ara, la 'Finestra' és una classe de la que hereten les classes 'Finestra Xarxa Carreteres' i 'Finestra Parametrització'. Aquest canvi és produeix perquè per a realitzar els càlculs de l'aplicació s'utilitza una parametrització de la xarxa de carreteres i així és possible visualitzar-la. Així doncs, existiran dos tipus de finestres a l'aplicació: les que mostraran xarxes de carreteres i les que contindran la parametrització de les primeres.

Abans de mostrar cadascuna de les classes detalladament s'explicarà a grans trets les relacions que existeixen entre elles:

1. La classe 'Finestra Principal' correspon a la finestra principal de l'aplicació. Aquesta contindrà una barra d'eines i un conjunt de finestres. El número de

finestres que contindrà serà un número parell, ja que per a cada xarxa de carreteres que mostrem, també s'obrirà una finestra amb la seva parametrització.

2. Cada parella de finestres (xarxa de carreteres i parametrització) contindrà un únic graf.

3. La classe 'Graf' és, sens dubte, la classe més important de l'aplicació. Aquesta conté les tres estructures de dades per a ordenar els vèrtexs, arestes i objectes de la xarxa. A més a més, ha de guardar els vehicles (ja que no poden formar part de cap estructura de dades) i calcular el seu moviment a través de les arestes. Finalment, la classe graf també és l'encarregada de resoldre tots els problemes de proximitat.

4. Les classes 'Facilitat', 'Punt de consulta' i 'Vehicle' hereten de la classe 'Objecte'. La classe 'Objecte' representa un objecte damunt la xarxa de carreteres, que pot ser un edifici, una persona o un vehicle. Els atributs més importants d'aquestes classes són la coordenada (longitud i latitud) i el seu punt de parametrització.

Cal dir que en aquest diagrama no s'han tingut en compte les classes menys importants, que estan relacionades amb la interfície i la interacció amb l'usuari. Aquestes classes són finestres emergents (diàlegs) que s'utilitzen tant per a mostrar el resultat d'un problema a l'usuari com per a demanar-li que introdueixi les dades amb les quals vol que es facin els càlculs.

4.2.1. Detall de les classes utilitzades

En aquest apartat s'explicaran totes les classes utilitzades en el diagrama de classes per poder-les entendre amb més claredat i poder veure quins són els seus atributs i mètodes. Només es detallaran els atributs i mètodes principals ja que hi ha classes que tenen una gran quantitat d'atributs i mètodes necessaris tan sols per a la implementació.

4.2.1.1. Classe: Finestra Principal

La classe Finestra Principal és l'encarregada de la interfície gràfica de l'aplicació. Permet fer visibles cadascun dels controls (menús, diàlegs, ...) que apareixen a la interfície del programa. Aquest conjunt d'elements gràfics han estat possibles gràcies a les llibreries que ens aporten les Qt4. Cadascun dels controls genera una sèrie d'events en funció del comportament de l'usuari en el cas que aquest modifiqui el seu estat. Aquests senyals normalment són enviats a la classe Finestra perquè dugui a terme les modificacions i els càlculs necessaris.

Els slots permeten respondre a una crida produïda. La majoria dels elements dissenyats dins la classe Finestra Principal es troben connectats a funcions del tipus slot que són executats en el moment que es produeix l'event indicat en la connexió.

Components de la classe:

- Atributs:
  - **BarraEines**: barra d'eines.
  - **Escriptori**: zona principal de la interfície on es situen totes les finestres.
  - **Menús**: diferents menús podem trobar al menú principal.
- Mètodes:
  - **Constructor**: permet crear l'objecte FinestraPrincipal.
  - **Destructor**. Permet destruir l'objecte FinestraPrincipal.
  - **Setups**: permeten crear cada un dels diferents elements.
  - **Slots**: reben el senyal produït pels botons del menú principal i de la barra d'eines.

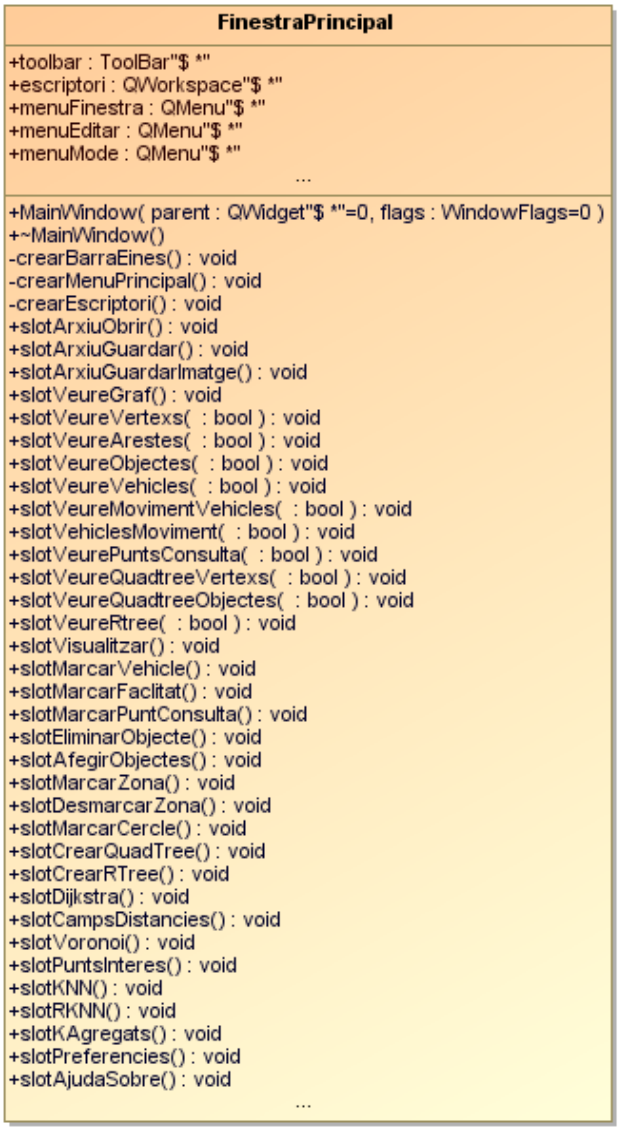


Figura 4.16: Classe: Finestra Principal

4.2.1.2. Classe: BarraEines

La classe BarraEines és l'encarregada de crear una barra d'eines on es mostren totes les accions més habituals que pot realitzar l'usuari.

Components de la classe:

- Atributs:
  - **Botons**: tenim cadascun dels botons col·locats a la barar d'eines.
- Mètodes:
  - **Constructor**: permet crear l'objecte BarraEines.
  - **Destructor**: permet destruir l'objecte BarraEines.

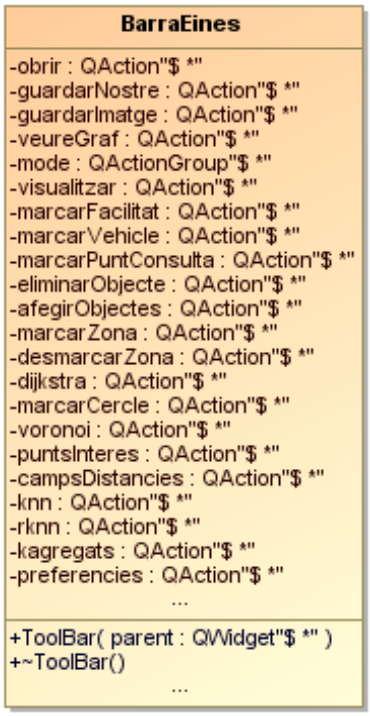


Figura 4.17: Classe: BarraEines

4.2.1.3. Classe: Finestra

La classe Finestra pot ser de dos tipus diferents segons el què s'hagi de mostrar en ella. El primer tipus que tenim s'utilitza per a mostrar la xarxa de carreteres i, per tant, s'utilitza la classe Finestra Xarxa Carreteres. El segon tipus que tenim s'utilitza per a mostrar la parametrització de la xarxa i, per tant, s'utilitza la classe Finestra Parametrització.

Components de la classe:

- Atributs:
  - **Tipus**: indica si la finestra de visualització és del tipus Xarxa de carretes o Parametrització.
  - **Opengl**: finestra de visualització.
  - **NomFitxer**: nom de la xarxa de carreteres que conté la finestra, per a poder-lo mostrar.
- Mètodes:
  - **Constructor**: permet construir l'objecte Finestra.
  - **Destructor**: permet destruir l'objecte Finestra.
  - **getTipus**: retorna l'atribut tipus.

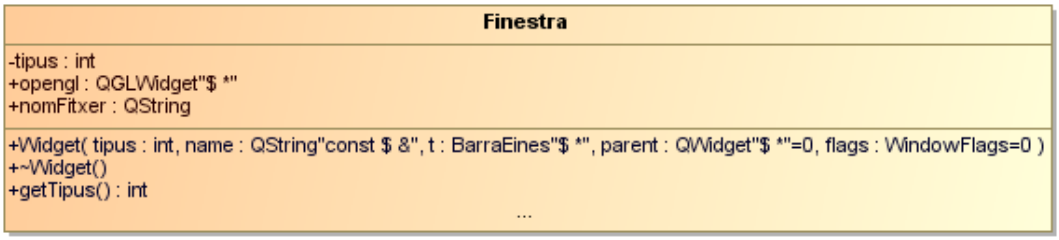


Figura 4.18: Classe: Finestra

4.2.1.4. Classe: Finestra Xarxa Carreteres

La classe Finestra Xarxa Carreteres hereta de la classe Finestra. Aquesta classe permet la visualització de la xarxa de carreteres així com dels objectes i dels resultats dels càlculs fets. A més, aquesta classe també s'encarrega d'interactuar amb l'usuari.

Components de la classe:

- Atributs:
  - **Mode**: indica si la fines
  - **GrafMapa:** graf de la xarxa de carreteres.
  - **GrafZona:** si existeix, és un subgraf del grafMapa.
  - **Indicadors de visualització:** conjunt de booleans que ens indiquen què cal visualitzar cada cop que es crida el mètode paintGL().

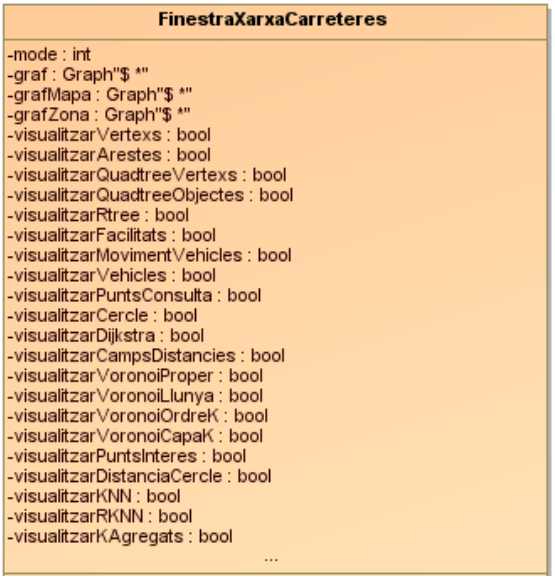


Figura 4.19: Classe: Finestra Xarxa Carreteres: Atributs

- Mètodes:
  - **TransformarCoordenades:** permet transformar un punt de pantalla en una coordenada de la xarxa de carreteres.
  - **TransformarPixelsADistancia:** permet transformar un determinat nombre de píxels en la distància equivalent al zoom en què en visualitza la xarxa.
  - **Mètodes bàsics visualització OpenGL:** mètodes de l'OpenGL necessaris per a la visualització.
  - **Constructor:** permet construir l'objecte Finestra Xarxa Carreteres.
  - **Destructor:** permet destruir l'objecte Finestra Xarxa Carreteres.
  - **Sets:** permeten canviar el valors dels atributs.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **CrearQuadTree:** mètode que s'encarrega de la creació del QuadTree.
  - **CrearRTree:** mètode que s'encarrega de la creació del RTree.
  - **AlgorismeDijkstra:** mètode que s'encarrega d'aplicar l'algorisme de Dijkstra.
  - **CampsDistancies:** mètode que s'encarrega de generar els camps de distàncies.
  - **DiagramaVoronoi:** mètode que s'encarrega de generar el diagrama de Voronoi.
  - **PuntsInteres**: mètode que s'encarrega de calcular els punts d'interès.
  - **AfegirObjectes**: mètode que s'encarrega d'afegir els objectes de forma aleatòria.
  - **DesmarcarZona**: mètode que s'encarrega de sortir de la zona marcada i tornar al graf original.
  - **CalcularCercle**: mètode que s'encarrega de calcular el cercle.
  - **KNN**: mètode que s'encarrega de resoldre el problema k-NN.
  - **RKNN:** mètode que s'encarrega de resoldre el problema Rk-NN.
  - **KAgregats**: mètode que s'encarrega de resoldre el problema Ak-NN.
  - **GuardarImatge**: mètode que permet generar una imatge de la finestra que s'està visualitzant.
  - **Preferències**: mètode que permet gestionar les preferències de l'usuari.
  - **Veure**: mètodes que permeten modificar els indicadors de visualització.

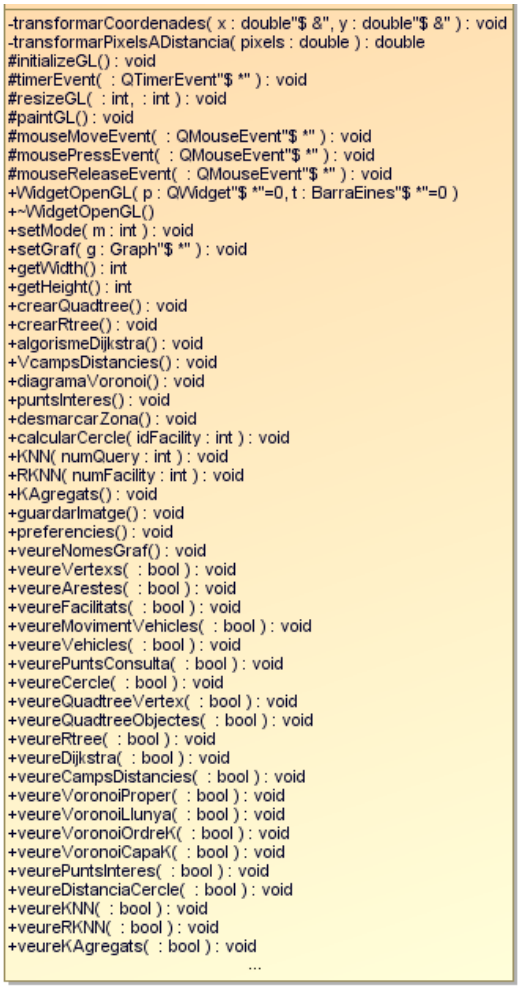


Figura 4.20: Classe: Finestra Xarxa Carreteres: Mètodes

4.2.1.5. Classe: Finestra Parametrització

La funció de la classe Finestra Parametrització és poder visualitzar els càlculs fets sobre la parametrització de la xarxa. És una classe de la qual podríem prescindir ja que els resultats obtinguts sobre la parametrització després es visualitzen sobre la xarxa de carreteres.

Components de la classe:

- Atributs:
  - **Graf**: conté el graf amb el qual s'està treballant en el Finestra Xarxa de carreteres. Aquest graf pot ser el graf del mapa o el graf de la zona.
  - **Indicadors de visualització**: conjunt de booleans que ens indiquen què cal visualitzar cada cop que es crida el mètode paintGL().
- Mètodes**:**
  - **Mètodes bàsics visualització OpenGL:** mètodes de l'OpenGL necessaris per a la visualització.

- **Constructor**: permet construir un objecte Finestra Parametrització.
- **Destructor**: permet destruir un objecte Finestra Parametrització.
- **setGraf**: permet modificar l'atribut graf.
- **Pintar**: mètodes que permet visualitzar diferents càlculs.

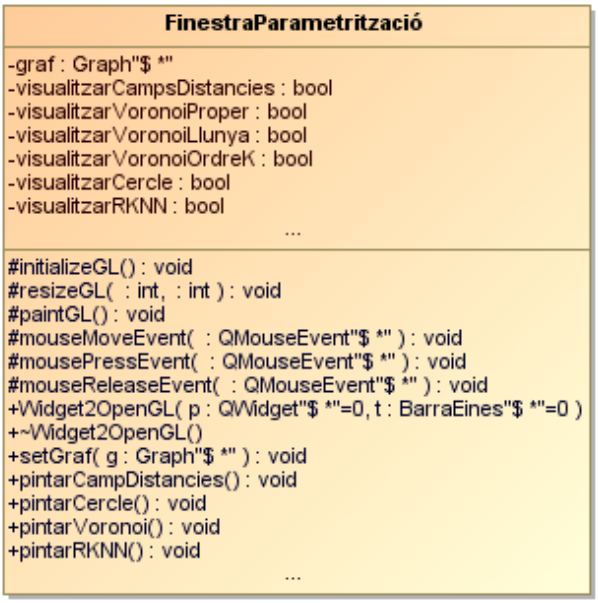


Figura 4.21: Classe: Finestra Parametrització

4.2.1.6. Classe: Graf

La classe Graf és la classe central de l'aplicació. Conté la xarxa de carreteres i tots els objectes necessaris per a poder resoldre els diferents problemes de proximitat.

Components de la classe:

- Atributs:
  - **WidthFinestra**: indica l'amplada que s'utilitza alhora de realitzar els càlculs, que per defecte serà l'amplada de la finestra.
  - **HeightFinestra**: indica l'alçada que s'utilitza alhora de realitzar els càlculs, que per defecte serà l'alçada de la finestra.
  - **LlistaVèrtexs**: vector que conté tots els vèrtexs del graf.
  - **LlistaArestes**: vector que conté totes les arestes del graf.
  - **ObjecteQuadTree**: QuadTree de facilitats i punts de consulta de la xarxa de carreteres.
  - **VèrtexQuadTree**: QuadTree de vèrtexs del graf.
  - **Rtree**: Rtree d'arestes del graf.
  - **Vehicles**: vector de vehicles que es troben a la xarxa.
  - **VehiclesEnMoviment**: booleà que ens indica si els vehicles tenen el moviment activat o desactivat.

- **Tex**: estructura que conté les variables necessàries per a poder treballar amb textures.
- **Gh**: estructura que conté les variables necessàries per a poder utilitzar el hardware gràfic.
- **Fbo**: frame buffer per a poder fer els càlculs de forma independent.
- **Variables de resultats**: conjunt de variables i vectors d'aquestes que ens permeten guardar els diferents resultats obtinguts.

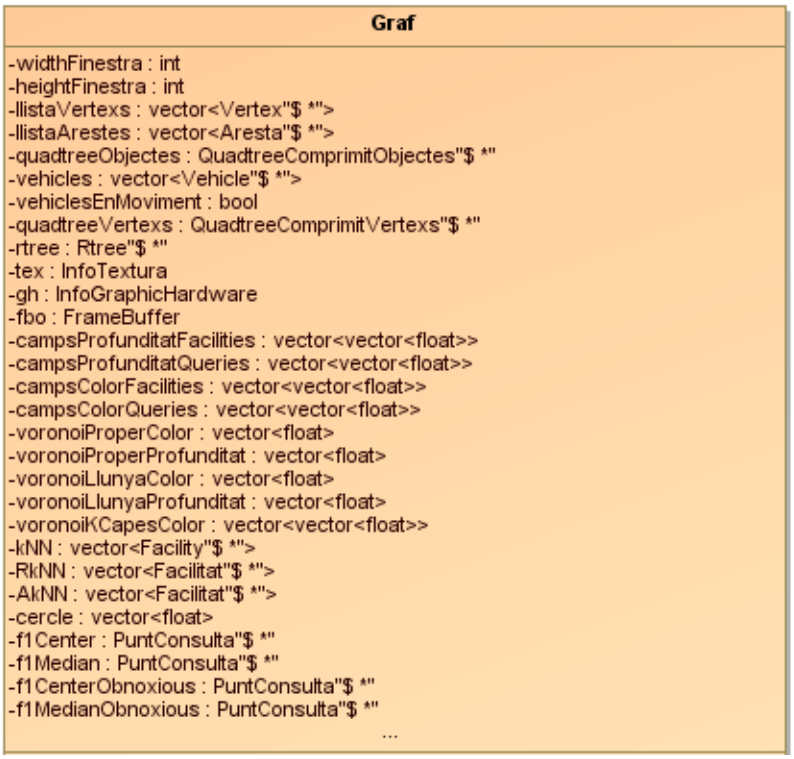


Figura 4.22: Classe: Graf: Atributs

- Mètodes:
  - **MoureVehicles:** permet activar o desactivar el moviment dels vehicles.
  - **ChooseProfiles:** mètode per a seleccionar un fragment profile.
  - **AssignarColorFacilitats:** permet assignar un color diferent a cada una de les facilitats de la xarxa.
  - **MètodesFBO:** permeten tractar el frame buffer.
  - **Constructors:** permet construir un objecte Graf.
  - **Destructor:** permet destruir un objecte Graf.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **AfegirObjectes:** afegeix objectes de forma aleatòria.
  - **DeleteObjectes:** permet eliminar una facilitat o un punt de consulta de la xarxa.
  - **DeleteVehicle**: permet eliminar un vehicle de la xarxa.
  - **SetObjectes**: permet canviar els objectes de la xarxa de carreteres (facilitats i punts de consulta).

- **SetVehicles**: permet canviar els vehicles de la xarxa de carreteres.
- **SetVehiclesEnMoviment**: permet activar o desactivar el moviment del vehicles.
- **MarcarFacilitat**: permet afegir una facilitat a la xarxa.
- **MarcarVehicle**: permet afegir un vehicle a la xarxa.
- **MarcarPuntConsulta**: permet afegir un punt de consulta a la xarxa.
- **CrearQuadTree**: mètode que permet crear un QuadTree.
- **CrearRTree**: mètode que permet crear un Rtree.
- **CalcularParametrització**: mètode per a calcular la parametrització de la xarxa.
- **AlgorismeDijkstra**: mètode que propaga l'algorisme de Dijkstra.
- **CalcularCampsDistancies**: mètode que calcula la funció camp de distàncies.
- **CalcularVoronoi**: mètode que genera els diagrames de Voronoi proper i llunyà.
- **CalcularVoronoiK**: mètode que genera el diagrama de Voronoi d'ordre k.
- **CalcularPuntsInteres**: mètode que calcular els punts d'interès de la xarxa.
- **CalcularCercle**: mètode que calcula el cercle.
- **CalcularKNN**: mètode que resol el problema k-NN.
- **CalcularAKNN**: mètode que resol el problem Ak-NN.
- **CalcularRKNN**: mètode que resol el problema Rk-NN.
- **PintarParametrització**: mètode per a visualitzar la parametrització de la xarxa.
- **Pintar**: mètode per a visualitzar el graf.

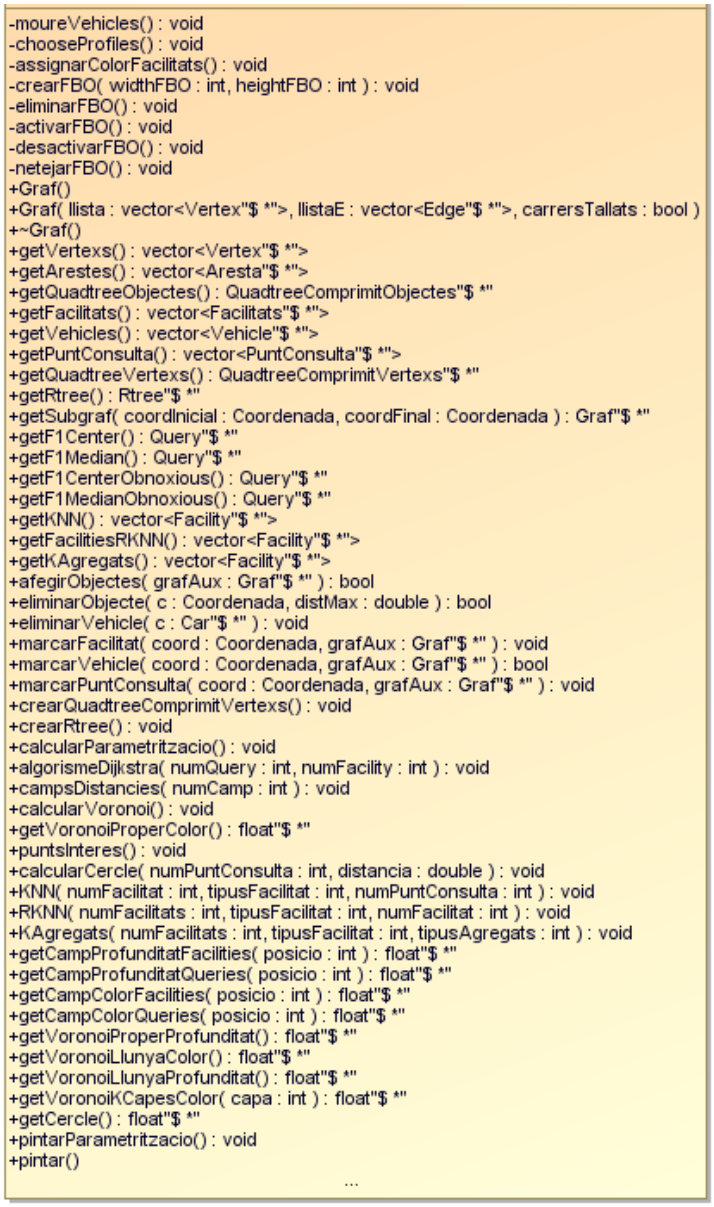


Figura 4.23: Classe: Graf: Mètodes

4.2.1.7. Classe: Vèrtex

La classe Vèrtex és necessària per a poder construir el Graf i representa part d'una carretera.

Components de la classe:

- Atributs:
  - **Id**: valor que identifica el vèrtex.
  - **LlistaArestes**: vector de referències a totes les arestes que té el vèrtex.
  - **Coordenada**: posició del vèrtex sobre la xarxa de carreteres.

- **vAnt**: atribut necessari per tal de calcular el camí mínim. Contindrà una referència al vèrtex antecessor en el recorregut del camí.
- **Cost camí mínim**: atribut necessari per tal de calcular el camí mínim. Contindrà el valor del cost mínim necessari per arribar fins a aquest vèrtex.

Mètodes:

- **Constructors**: permeten crear objectes Vèrtex.
- **Destructors**: permet destruir un objecte Vèrtex.
- **Gets:** retornen atributs o informació de l'objecte.
- **Sets:** permeten canviar el valors dels atributs.
- **CalcularGrafConnex**: mètode per a obtenir el graf connex amb nombre de vèrtexs major.
- **AddAresta**: permet afegir una aresta al vèrtex.
- **CarrersTallats**: mètode per marcar les arestes del vèrtex que representen carrers tallats.
- **Pintar**: mètode per a visualitzar el vèrtex.

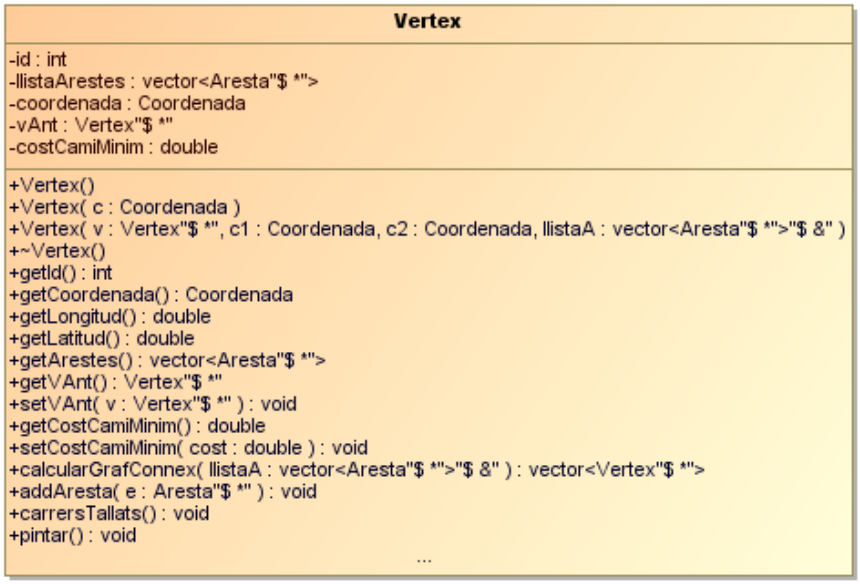


Figura 4.24: Classe: Vèrtex

4.2.1.8. Classe: Aresta

La classe Aresta és necessària per a poder construir el Graf i representa part d'una carretera.

Components de la classe:

Atributs:

- **Origen**: vèrtex on comença l'aresta.
- **Destí**: vèrtex on acaba l'aresta.

- **Longitud**: ens indica la longitud que té l'aresta i ens serveix per a resoldre per exemple els problemes de camins mínims on s'utilitza com a cost.
- **Categoria**: aquest atribut pot prendre diferents valors: autopista, nacional, comarcal, local, rural, culdesac o noclassificat. Segons la categoria sabem de quin tipus de carretera es tracta i podem visualitzar-la d'una forma o altra.
- **CarrerTallat**: booleà que indica si l'aresta és una aresta sense sortida.
- **XMin, XMax, YMin, YMax**: punts que generen el rectangle englobant que delimita l'aresta.
- **PInici ,PFinal**: punts de parametrització que ens indiquen la situació de l'aresta dins la parametrització de la xarxa.
- **ColorR, ColorG, ColorB**: color amb què es pinta l'aresta quan es representa la parametrització.

Mètodes:

- **Constructors**: permeten construir objectes Aresta.
- **Destructor**: permet destruir un objecte Aresta.
- **Gets:** retornen atributs o informació de l'objecte.
- **Sets:** permeten canviar el valors dels atributs.
- **FormaPart**: permet saber si una coordenada forma part del rectangle englobant de l'aresta.
- **CalcularPuntProper**: permet calcular la coordenada exacta sobre una aresta corresponent a una coordenada que es troba propera a l'aresta.
- **Pintar**: mètode per a visualitzar l'aresta.

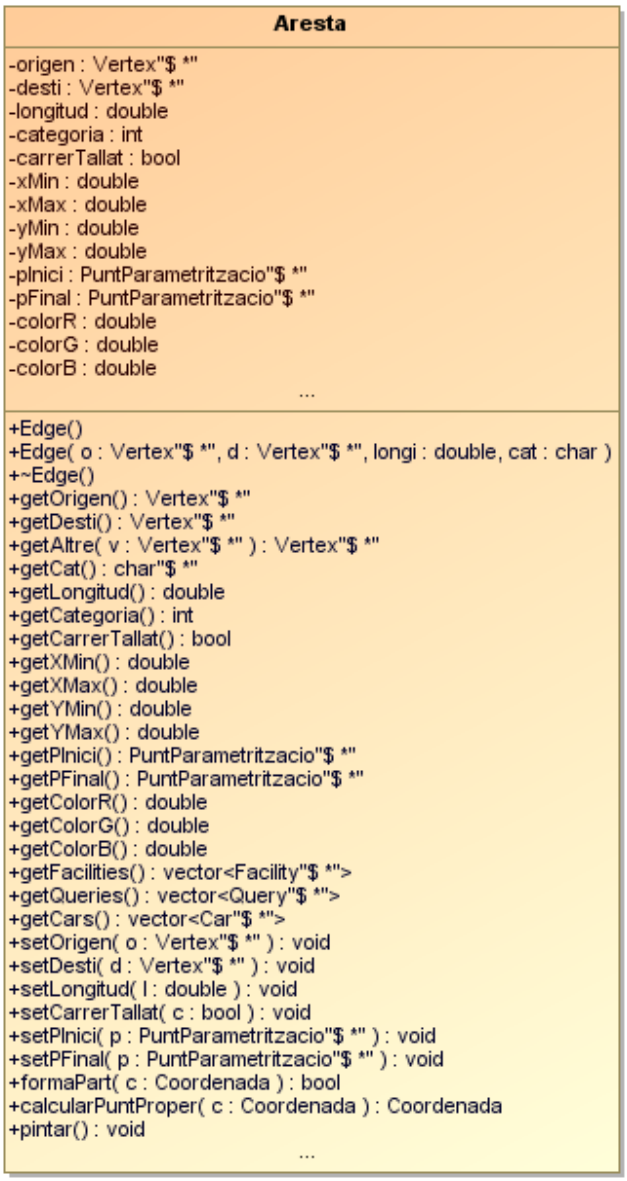


Figura 4.25: Classe: Aresta

4.2.1.9. Classe: Coordenada

La classe Coordenada és una classe bàsica encarregada de guardar cada una de les localitzacions de la xarxa de carreteres. Cada coordenada està formada per dos valors: longitud i latitud, que ens indiquen la situació dins la xarxa d'un determinat element com podrien ser els vèrtexs o els objectes.

Components de la classe:

- Atributs:
  - **Longitud**: valor de la component X expressat en coordenades cartesianes.

- **Latitud**: valor de la component Y expressat en coordenades cartesianes.

- Mètodes:
  - **Constructors**: permeten construir objectes Coordenada.
  - **Destructor**: permet destruir un objecte Coordenada.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **Dist**: retorna la distància entre la coordenada actual i la passada per paràmetre.

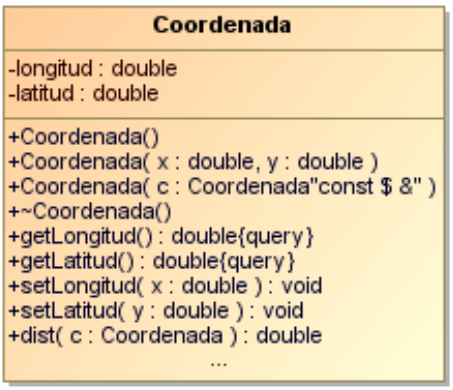


Figura 4.26: Classe: Coordenada

4.2.1.10. Classe: Punt Parametrització

La classe Punt Parametrització és conceptualment molt similar a la classe Coordenada que hem vist anteriorment. La diferència és que ens indica una posició dins la parametrització de la xarxa enlloc d'indicar-nos una localització dins la xarxa de carreteres. Tots els vèrtexs i objectes tenen un objecte d'aquesta classe que permet accedir ràpidament a la seva posició dins la parametrització.

Components de la classe:

- Atributs:
  - **X**: valor de la component X que indica una posició dins la parametrització de la xarxa.
  - **Y**: valor de la component Y que indica una posició dins la parametrització de la xarxa.
- Mètodes:
  - **Constructors**: permeten construir objectes PuntParametrització.
  - **Destructor**: permet destruir un objecte PuntParametrització.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.

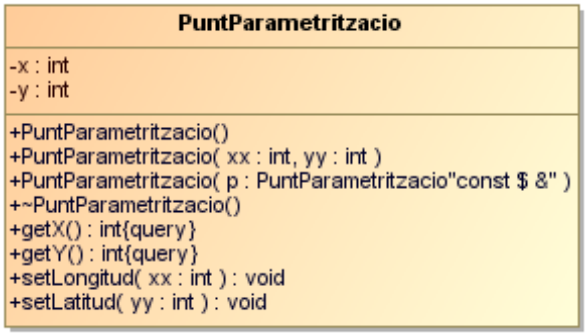


Figura 4.27: Classe: Punt Parametrització

4.2.1.11. Classe: Objecte

Aquesta classe defineix els atributs bàsics que necessiten les facilitats, els punts de consulta i els vehicles. No s'utilitza per crear instàncies 'Objecte' ja que sempre es creen objectes específics d'una de les 3 classes que hereten d'ella, però si que ens serveix per a tractar les facilitats, els punts de consulta i els vehicles com si fossin un mateix tipus d'objectes, per exemple, per a ordenar-los a les estructures de dades.

Components de la classe:

- Atributs:
  - **Coordenada**: per localitzar l'objecte dins la xarxa de carreteres
  - **PosicióParametrització**: punt de parametrització per a localitzar l'objecte dins la parametrització de la xarxa.
  - **Aresta**: ens indica de quina aresta forma part l'objecte.
  - **ColorR, ColorG, ColorB**: per a resoldre alguns problemes de proximitat és necessari que cada objecte tingui assignat un color diferent.
- Mètodes:
  - **Constructors**: permeten construir instàncies Objecte.
  - **Destructor**: permet destruir una instància Objecte.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **Pintar**: permet visualitzar un objecte.

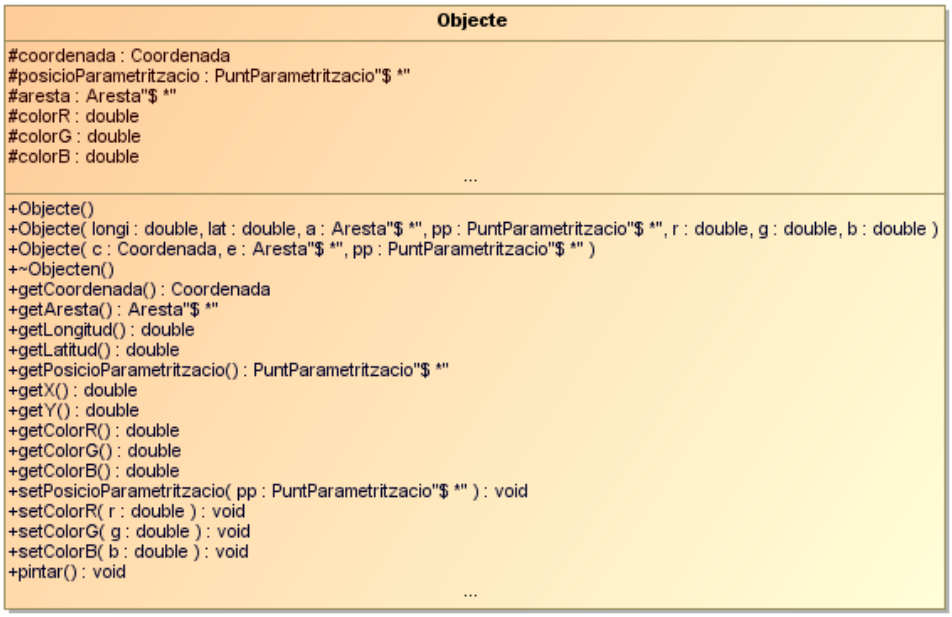


Figura 4.28: Classe: Objecte

4.2.1.12. Classe: Facilitat

La classe Facilitat hereta de la classe Objecte i representa un edifici de la xarxa. A més dels atributs que hereta de la Objecte té alguns atributs específics per tal d'identificar-la: el nom, l'adreça, la pàgina web i el tipus. El tipus indica la categoria en què es classifica la facilitat. S'han introduït 6 tipus bàsics: gasolinera, restaurant, hotel, hospital, central nuclear i aeroport. També s'ha introduït un setè tipus per totes aquelles facilitats que no es poden classificar en cap dels anteriors.

Components de la classe:

- Atributs:
  - **Nom**: nom que identifica la facilitat.
  - **Adreça**: adreça on s'ubica la facilitat.
  - **Web**: pàgina web que té la facilitat.
  - **Tipus**: tipus de la facilitat (hotel, restaurant, etc.)
- Mètodes:
  - **Constructors**: permeten construir objectes Facilitat.
  - **Destructor**: permet destruir un objecte Facilitat.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **PintarInfo**: permet visualitzar la informació de la facilitat.

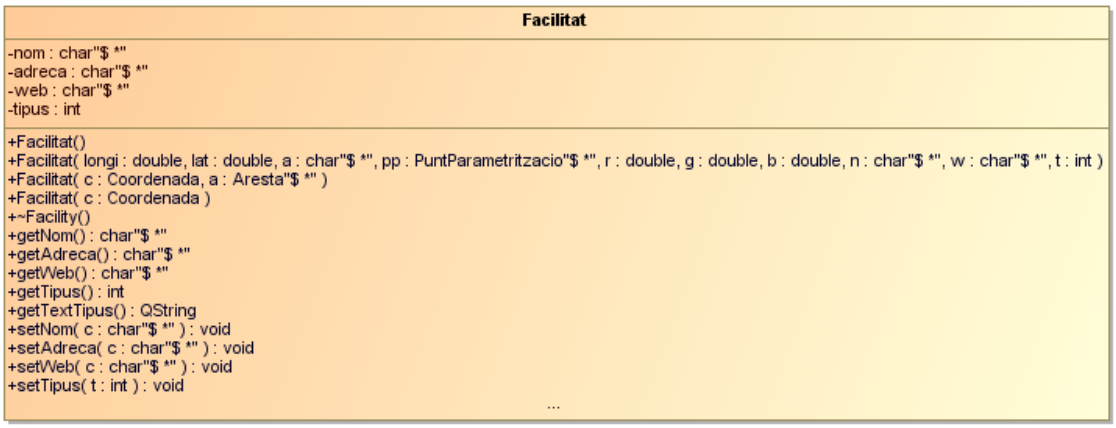


Figura 4.29: Classe: Facilitat

4.2.1.13. Classe: Punt de consulta

La classe Punt de consulta hereta de la classe Objecte i representa una persona en el sistema. L'únic atribut que té específic és el nom amb el qual el podem identificar.

Components de la classe:

- Atributs:
  - **Nom**: nom que identifica el punt de consulta.
- Mètodes:
  - **Constructors**: permeten construir objectes Punt de consulta.
  - **Destructor**: permet destruir un objecte Punt de consulta.
  - **GetNom:** retornen l'atribut nom.
  - **SetNom:** permeten canviar l'atribut nom.

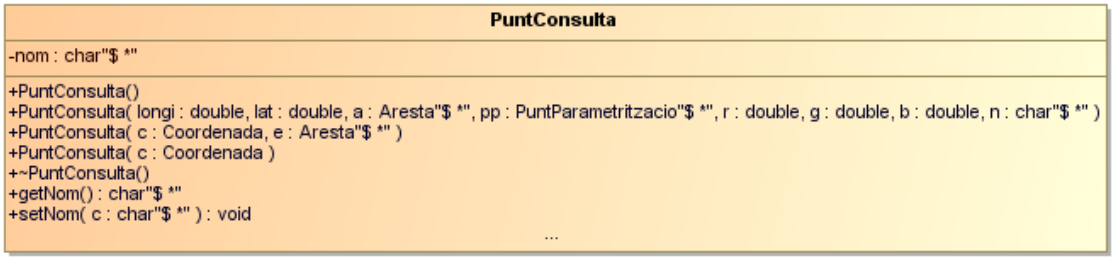


Figura 4.30: Classe: Punt de consulta

4.2.1.14. Classe: Vehicle

La classe vehicle hereta de la classe Objecte i representa un vehicle de la xarxa.

Components de la classe:

- Atributs:
  - **Nom**: ens indica el nom amb el qual el podem identificar, que podria ser per exemple la matrícula del vehicle.
  - **Velocitat**: ens indica la velocitat a la qual circula. Aquesta velocitat pot prendre cinc valors diferents: molt ràpid, ràpid, normal, lent o molt lent.
  - **Sentit**: ens indica cap a quin vèrtex es dirigeix.
- Mètodes:
  - **Constructors**: permeten construir objectes Vehicle.
  - **Destructor**: permet destruir un objecte Vehicle.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **Moure:** genera el moviment del car.
  - **ActualitzarSentit**: permet assignar el sentit del vehicle dins l'aresta.
  - **PintarInfo**: permet visualitzar la informació del car

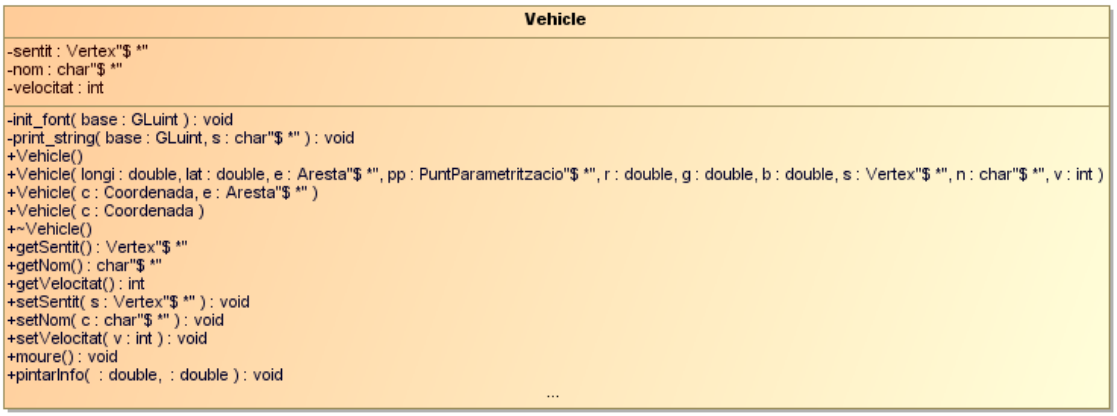


Figura 4.31: Classe: Vehicle

4.2.1.15. Classe: QuadTree Comprimit de Vèrtexs

La classe QuadTree Comprimit de Vèrtexs conté els atributs i mètodes necessaris per a crear una estructura de dades en arbre per a ordenar un conjunt de vèrtexs distribuïts en un pla 2D.

Components de la classe:

- Atributs:
  - **LlistaVèrtexs**: vector que conté els vèrtexs del graf.
  - **Level**: nivell actual de l'arbre.

- **MaxLevel**: nivell màxim que pot arribar a tenir.
- **Root**: node arrel de l'arbre.
- **Length**: mida del quadrat englobant del QuadTree.
- **BasePoint**: Cantonada superior esquerra del quadrat englobant.

Mètodes:

- **CreateBox**: crea la caixa englobant.
- **Constructors**: permeten construir objectes QuadTree Comprimit de Vèrtexs.
- **Destructor**: permet destruir un objecte QuadTree Comprimit de Vèrtexs.
- **Gets:** retornen atributs o informació de l'objecte.
- **AddVèrtex:** afegeix un vèrtex al QuadTree.
- **SearchVèrtex:** busca un vèrtex dins el QuadTree.
- **CalcularGraf:** crea un subgraf a partir d'una zona rectangular.
- **Pintar**: permet visualitzar l'estructura del QuadTree.

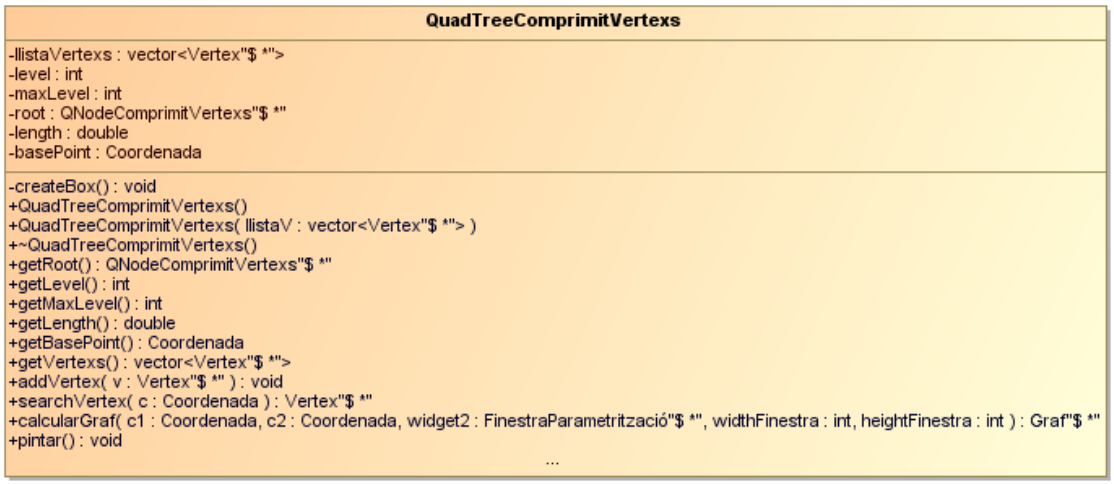


Figura 4.32: Classe: QuadTree Comprimit de Vèrtexs

4.2.1.16. Classe: QNode Comprimit de Vèrtexs

La classe QNode Comprimit de Vèrtexs representa un node de l'estructura de dades QuadTree Comprimit de Vèrtexs.

Cada node contindrà el seu propi punt d'ancoratge i mida, així com la llista de vèrtexs que conté. A més, degut a la seva estructura en forma d'arbre, cada node tindrà 4 fills.

Components de la classe:

- Atributs:
  - **Length**: mida del quadrat englobant del node.
  - **Level**: nivell del node.
  - **BasePoint**: cantonada superior esquerra del quadrat englobant del node
  - **LlistaVèrtexs**: vector que conté els vèrtexs del node.
  - **S1, S2, S3 i S4**: fills del node.
- Mètodes:
  - **Constructors**: permeten construir objectes QNode Comprimit de Vèrtexs.
  - **Destructor**: permet destruir un objecte QNode Comprimit de Vèrtexs.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **AddVèrtex:** afegeix un vèrtex al node.
  - **SearchVèrtex:** busca un vèrtex en el node.
  - **CalcularGraf:** retorna els vèrtexs continguts en la zona rectangular.
  - **Contained:** indica si el node està dins la zona rectangular.
  - **Stabbling:** indica si una part del node està dins la zona rectangular.
  - **IsLeaf:** indica si el node és fulla negra, blanca o node parcial.
  - **InWhatSon:** retorna el número de fill que correspon al paràmetre passat.
  - **CreateSons:** crea els fills del node.
  - **AddToCorrespondingSon:** afegeix el vèrtex al fill corresponent.
  - **InNodePuntConsulta:** indica si el vèrtex forma part del node.
  - **SonNotWhiteLeaf:** indica el número de fill que no és fulla blanca. Si hi ha més d'un fill no fulla blanca retorna 0.
  - **Pintar:** permet visualitzar el node.

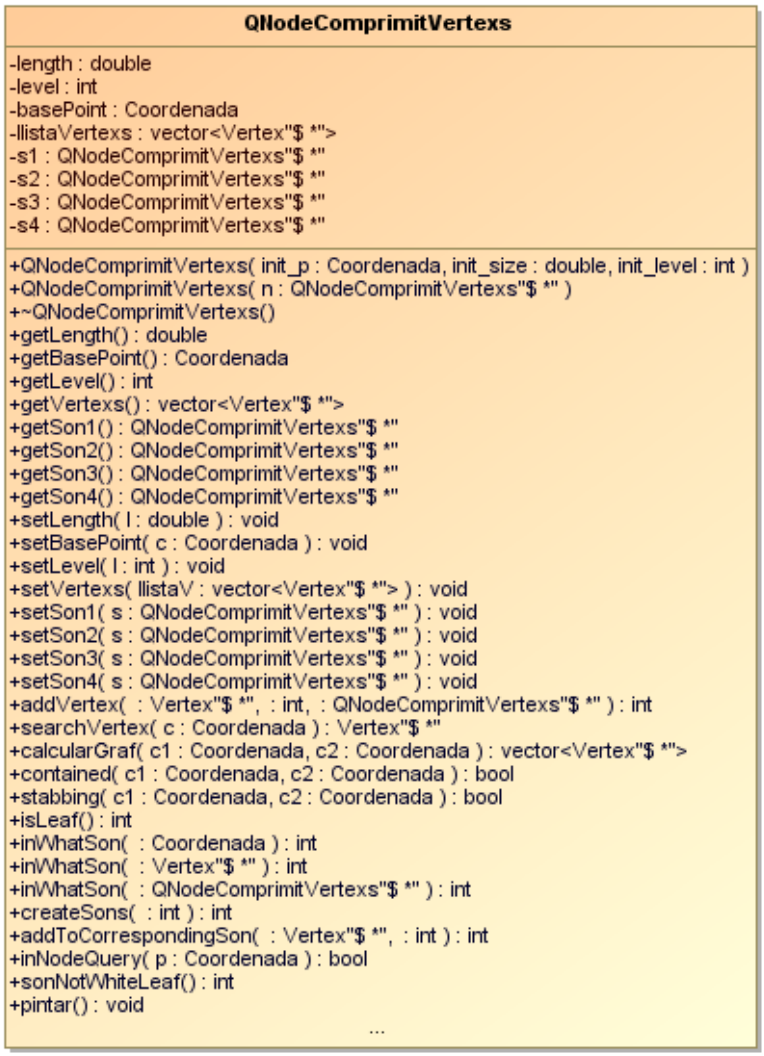


Figura 4.33: Classe: QNode Comprimit de Vèrtexs

4.2.1.17. Classe: Rtree

La classe RTree conté els atributs i mètodes necessaris per a crear una estructura de dades en arbre per a ordenar un conjunt d'arestes distribuïts en un pla 2D.

Components de la classe:

- Atributs:
  - **LlistaArestes**: vector que conté les arestes del graf.
  - **Root**: arrel de l'arbre.
  - **XMin, XMax, YMin, YMax**: punts que limiten el rectangle englobant de les arestes del Rtree.
- Mètodes:
  - **CreateBox:** crea la caixa englobant.
  - **Constructors**: permeten construir objectes Facilitat.
  - **Destructor**: permet destruir un objecte Facilitat.

- **Gets:** retornen atributs o informació de l'objecte.
- **AddAresta**: afegeix una aresta al RTree.
- **SearchAresta**: busca una aresta al RTree.
- **Pintar**: permet visualitzar el RTree.

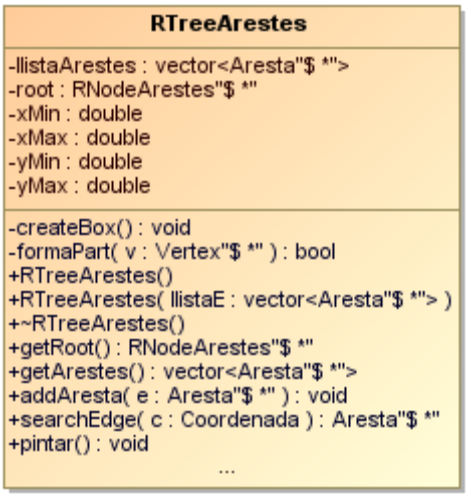


Figura 4.34: Classe: Rtree

4.2.1.18. Classe: Rnode

La classe RNode representa un node de l'estructura de dades RTree. Cada node contindrà el seu propi rectangle englobant, així com la llista d'arestes que conté. A més, degut a la seva estructura en forma d'arbre, cada node tindrà una llista de fills.

Components de la classe:

- Atributs:
  - **LlistaArestes**: vector que conté les arestes del node.
  - **XMin, XMax, YMin, YMax**: punts que limiten el rectangle englobant de les arestes del node.
  - **Level**: nivell del node.
  - **LlistaFills**: vector que conté una llista de nodes fills.
- Mètodes:
  - **Constructors**: permeten construir objectes Facilitat.
  - **Destructor**: permet destruir un objecte Facilitat.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **AddAresta:** afegeix una aresta al node.
  - **SearchAresta:** busca una aresta al node.
  - **IsLeaf:** retorna si un node és fulla blanca, negra o node parcial.
  - **InWhatSon:** retorna el número de fill que correspon al paràmetre passat.
  - **CalcularArea:** calcula l'àrea que formen el rectangle actual i la caixa englobant de l'aresta passada per paràmetre.

- **SplitNode:** permet dividir el node.
- **InNodePuntConsulta:** indica si la coordenada passada per paràmetre forma part del node.
- **Pintar**: permet visualitzar el node.

Figura 4.35: Classe: Rnode

4.2.1.19. Classe: QuadTree Comprimit d'Objectes

La classe QuadTree Comprimit d'Objectes conté els atributs i mètodes necessaris per a crear una estructura de dades en arbre per a ordenar un conjunt d'objectes (facilitats i punts de consulta) distribuïts en un pla 2D.

Components de la classe:

- Atributs:
  - **Facilitats**: vector que conté les facilitats del graf.
  - **PuntsConsulta**: vector que conté els punts de consulta del graf.
  - **Level**: nivell actual de l'arbre.
  - **MaxLevel**: nivell màxim que pot arribar a tenir.
  - **Root**: node arrel de l'arbre.
  - **Length**: mida del quadrat englobant del QuadTree.
  - **BasePoint**: Cantonada superior esquerra del quadrat englobant.

- Mètodes:
  - **CreateBox**: crea la caixa englobant.
  - **Reconstruir**: reconstrueix el QuadTree quan s'ha eliminat un objecte.
  - **Constructors**: permeten construir objectes QuadTree Comprimit de Vèrtexs.
  - **Destructor**: permet destruir un objecte QuadTree Comprimit de Vèrtexs.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **AddFacilitat:** afegeix una facilitat al QuadTree.
  - **AddPuntConsulta:** afegeix un punt de consulta al QuadTree.
  - **DeleteFacilitat:** elimina una facilitat del QuadTree.
  - **DeletePuntConsulta:** elimina un punt de consulta del QuadTree.
  - **SearchObjecte:** busca un objecte dins el QuadTree.
  - **SearchFacilitat:** busca una facilitat dins el QuadTree.
  - **SearchPuntConsulta:** busca un punt de consulta dins el QuadTree.
  - **Pintar**: permet visualitzar l'estructura del QuadTree.

Figura 4.36: Classe: QuadTree Comprimit d'Objectes

4.2.1.20. Classe: QNode Comprimit d'Objectes

La classe QNode Comprimit d'Objectes representa un node de l'estructura de dades QuadTree Comprimit d'Objectes.

Cada node contindrà el seu propi punt d'ancoratge i mida, així com la llista de facilitats i punts de consulta que conté. A més, degut a la seva estructura en forma d'arbre, cada node tindrà 4 fills.

Components de la classe:

- Atributs:
  - **Length**: mida del quadrat englobant del node.
  - **Level**: nivell del node.
  - **BasePoint**: cantonada superior esquerra del quadrat englobant del node.
  - **Facilitats**: vector que conté les facilitats del graf.
  - **PuntsConsulta**: vector que conté els punts de consulta del graf.
  - **S1, S2, S3 i S4**: fills del node.
- Mètodes:
  - **Constructors**: permeten construir objectes Vèrtex QNode Comprimit.
  - **Destructor**: permet destruir un objecte Vèrtex QNode Comprimit.
  - **Gets:** retornen atributs o informació de l'objecte.
  - **Sets:** permeten canviar el valors dels atributs.
  - **AddFacilitat:** afegeix una facilitat al node.
  - **AddPuntconsulta:** afegeix un punt de consulta al node.
  - **DeleteFacilitat:** elimina una facilitat del node.
  - **DeletePuntConsulta:** elimina un punt de consulta del node.
  - **SearchFacilitat:** busca una facilitat en el node.
  - **SearchPuntConsulta:** busca un punt de consulta en el node.
  - **IsLeaf:** indica si el node és fulla negra, blanca o node parcial.
  - **InWhatSon:** retorna el número de fill que correspon al paràmetre passat.
  - **CreateSons:** crea els fills del node.
  - **AddToCorrespondingSon:** afegeix la facilitat o el punt de consulta al fill corresponent.
  - **InNodePuntConsulta:** indica si la coordenada forma part del node.
  - **SonNotWhiteLeaf:** indica el número de fill que no és fulla blanca. Si hi ha més d'un fill no fulla blanca retorna 0.
  - **Pintar:** permet visualitzar el node.

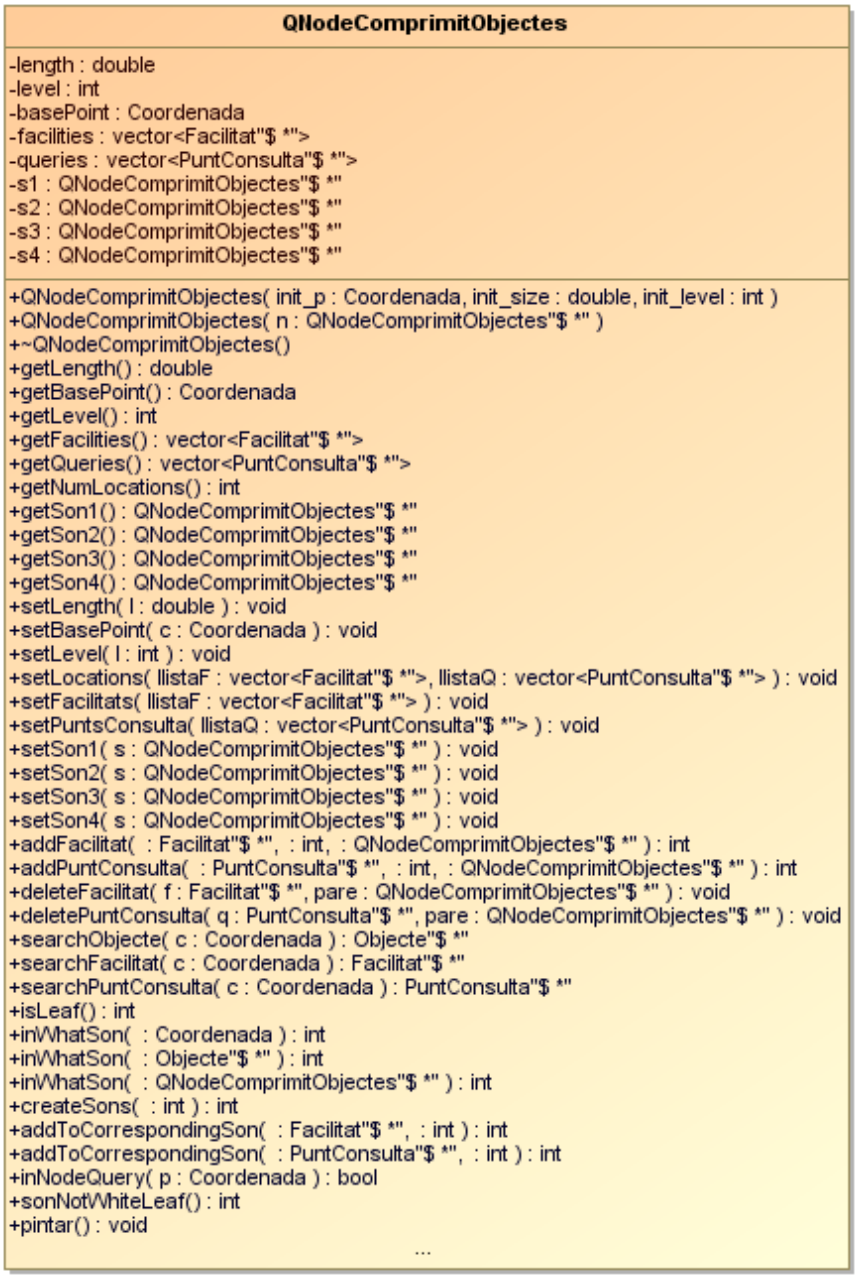


Figura 4.37: Classe: QNode Comprimit d'Objectes

4.3. Diagrames d'interacció

Els diagrames d'interacció són models dinàmics que descriuen com col·laboren els objectes que configuren un cas d'ús. Aquests diagrames mostren la interacció entre objectes, que ho fan enviant-se missatges. Es dibuixen amb el propòsit d'ajudar a decidir les operacions de cada classe.

Hi ha dos tipus de diagrames d'interacció:

- Diagrames de seqüència
- Diagrames de col·laboració

Aquests dos diagrames són equivalents, és a dir, proporcionen vistes alternatives de la mateixa informació.

Tot seguit es representen alguns dels diagrames de seqüència i col·laboració més importants en el sistema a implementar ja que ens permetrà lligar els diagrames anteriors.

4.3.1. Diagrames de seqüència

Un diagrama de seqüència és un diagrama d'interacció que detalla com s'executen les operacions en funció del temps, és a dir, quins missatges són enviats, per quin objecte, a quin objecte i quan.

4.3.1.1. Cas d'ús: Carregar Fitxer

Com ja hem comentat en els capítols anteriors, aquest cas d'ús s'utilitza per a carregar un fitxer de text i crear una xarxa de carreteres.

Observant el diagrama podem veure que, primerament, l'usuari ha de seleccionar quin fitxer vol carregar. Llavors es comprova que l'extensió sigui correcte i s'obre per a poder-lo llegir. Un cop s'ha llegit, es creen els vèrtexs i les arestes adjacents a ells.

Quan ja s'han creat tots els vèrtexs i arestes de la xarxa, s'ha de comprovar si existeix un fitxer d'objectes. En cas que existeixi, s'ha de llegir i anar creant tots els objectes (facilitats, punts de consulta i vehicles).

Un cop arribats en aquest punt, ja es té tot el necessari per a crear el graf i les estructures de dades dels seus objectes.

Finalment, s'han de crear les dues finestres: la primera per a visualitzar la xarxa de carreteres i la segona per a visualitzar la parametrització de la xarxa.

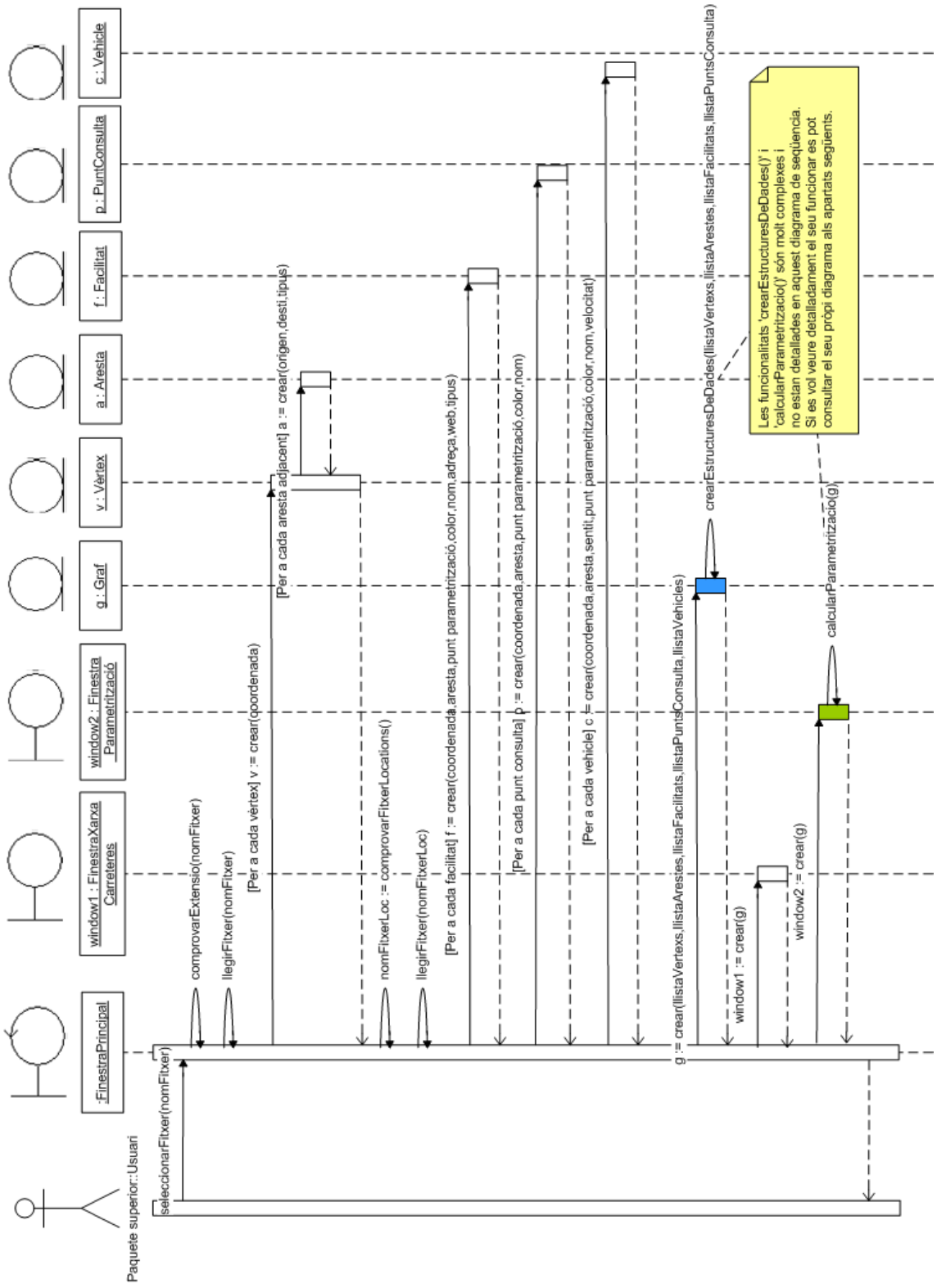


Figura 4.38: Diagrama de seqüència: Carregar fitxer

4.3.1.2. Cas d'ús: Afegir Facilitat

En aquest apartat es mostra el diagrama de seqüència del procés que ens permet afegir una facilitat a la xarxa de carreteres.

Per fer-ho, l'usuari ha de seleccionar un punt sobre la pantalla. Llavors, utilitzant l'estructura de dades Rtree es buscarà quin és l'aresta que es troba més a prop d'aquest punt. Si s'ha pogut trobar l'aresta, es calcularà quina és la coordenada exacta sobre ell que es troba més a prop del punt inicial.

Si s'ha aconseguit trobar l'aresta i la coordenada, l'usuari haurà d'entrar les dades per a la nova facilitat. Aquestes dades són el nom, l'adreça, el lloc web i un tipus dels que té predefinits l'aplicació.

Amb totes aquestes dades es crearà la nova facilitat i s'introduirà a la xarxa de carreteres mitjançant el QuadTree d'objectes.

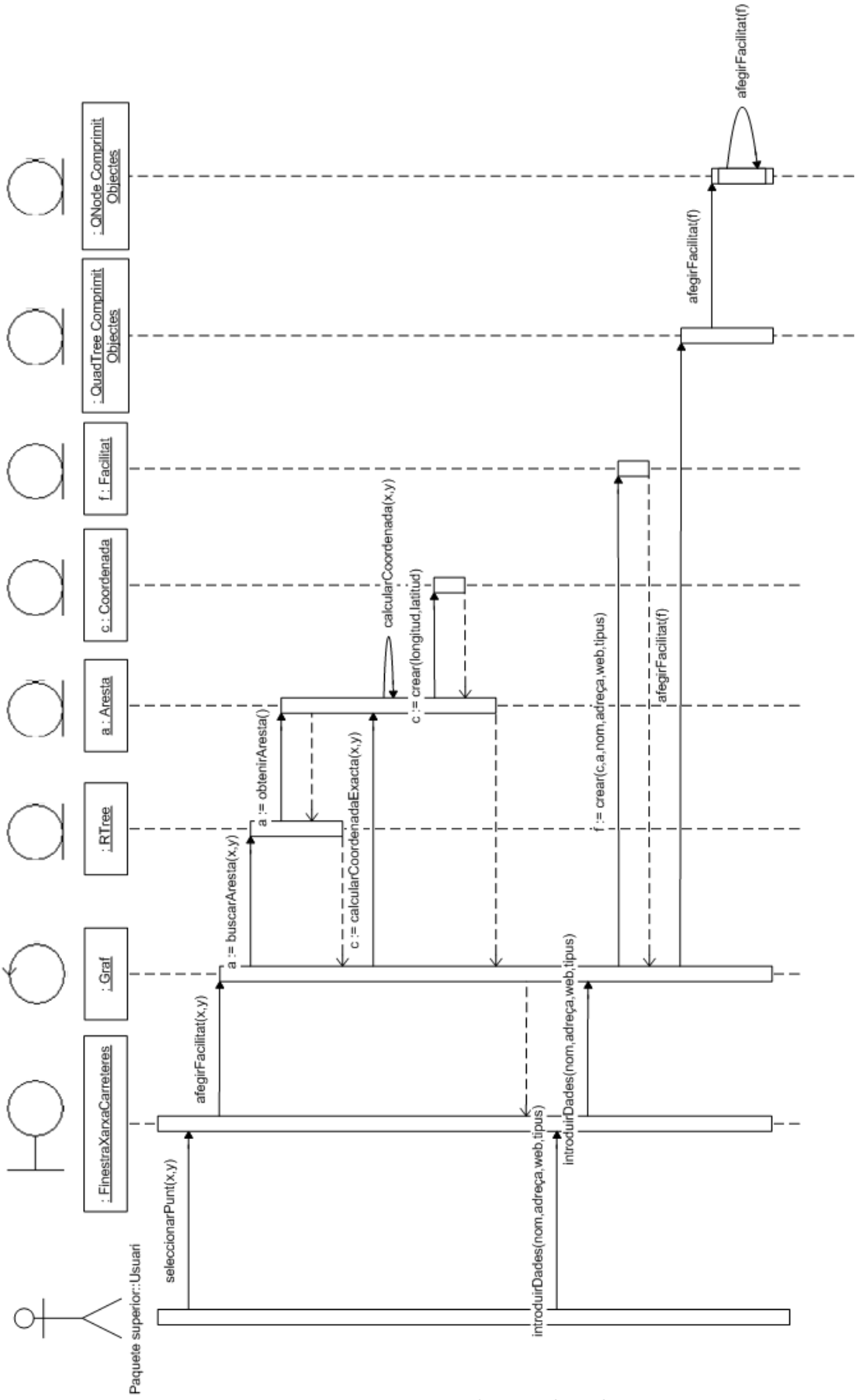


Figura 4.39: Diagrama de seqüència: Afegir facilitat

4.3.1.3. Cas d'ús: Crear estructures de dades

El diagrama de seqüència d'aquest cas d'ús hauria d'estar dins el diagrama carregar fitxer. El motiu pel qual s'ha separat és per la seva importància i complexitat.

Observant el diagrama, podem veure que les estructures de dades es creen al mateix temps que es crea el graf. La seva estructura està basada en arbres de dades i, per tant, el seu funcionament és similar.

Per a cada una de les estructures, el primer que es fa és calcular una caixa englobant que tindrà forma de quadrat en el cas dels QuadTree i forma rectangular en el cas dels Rtree. Aquesta caixa ha de el més petit possible i ha de contenir tots els objectes que formaran part de l'arbre.

Un cop s'ha calculat el punt d'ancoratge (cantonada superior-esquerra) i la mida de la caixa englobant, es crea el node arrel de l'arbre amb aquests dos atributs. A partir d'aquí es van introduint els objectes un per un dins el node arrel i aquest s'encarrega d'anar creant els fills i assignar-los-hi els objectes.

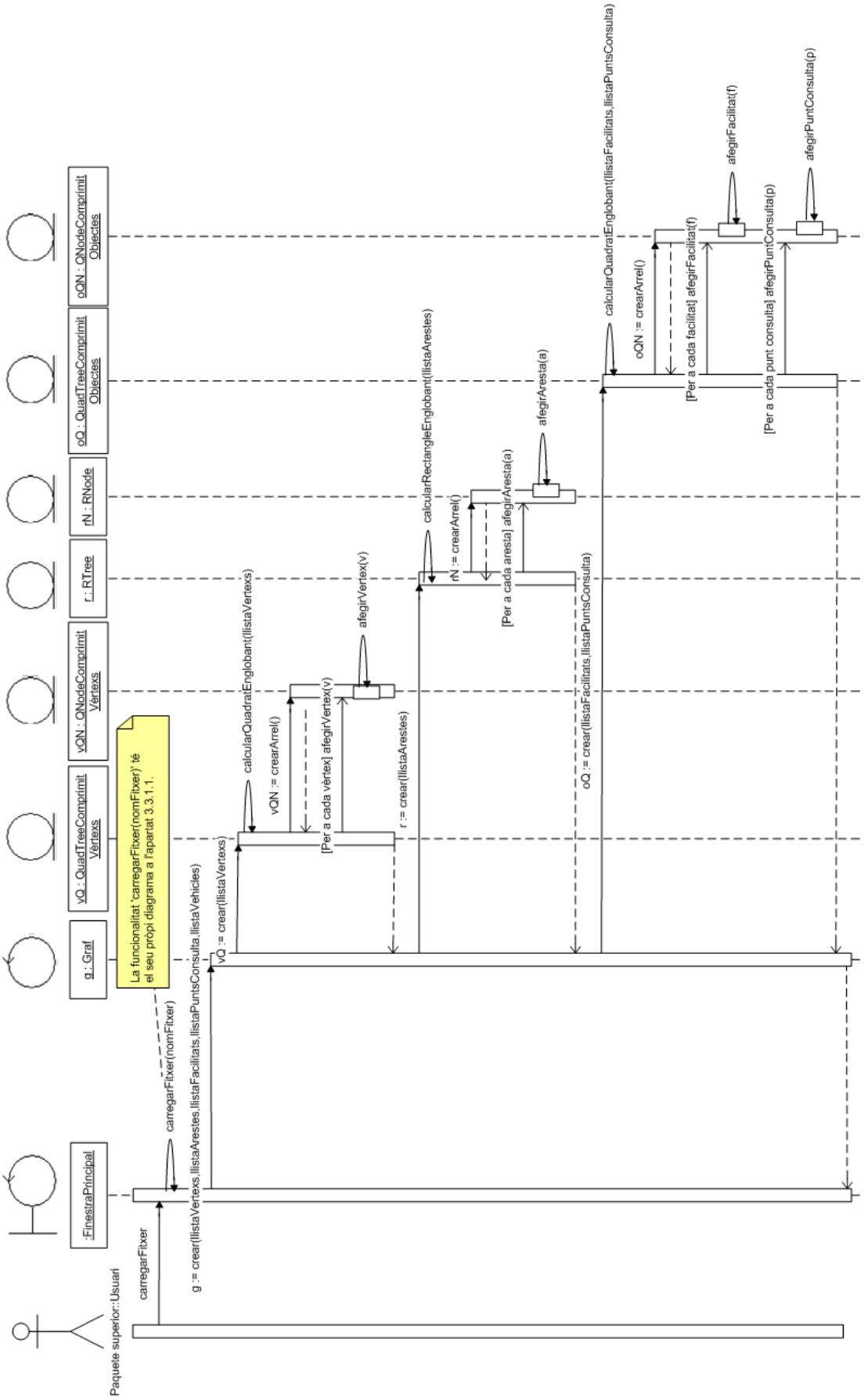


Figura 4.40: Diagrama de seqüència: Crear estructures de dades

4.3.2. Diagrames de col·laboració

Els diagrames de col·laboració proporcionen la mateixa informació que els diagrames de seqüència, però centrant-se en els rols dels objectes en comptes de fer-ho en el temps en el qual s'envien els missatges.

4.3.2.1. Cas d'ús: Seleccionar Objecte

En la següent figura es pot veure el diagrama de col·laboració corresponent al cas d'ús "Seleccionar Objecte". El seu funcionament consisteix en mostrar tota la informació d'un objecte seleccionat per l'usuari per pantalla.

Així doncs, l'usuari ha de seleccionar un punt sobre la pantalla i, utilitzant el QuadTree d'objectes, es farà una cerca per a intentar trobar un objecte a prop del punt.

Per a fer la cerca el QuadTree l'iniciarà pel node arrel de l'arbre i aquest, depenent del punt entrat per l'usuari, seguirà la cerca per un dels quatre fills. Aquest procés es repetirà fins a arribar a un node fulla que contindrà l'objecte que s'està buscant.

Finalment, es crearà un quadre de text just al costat de l'objecte on s'hi mostrarà tota la informació que conté.

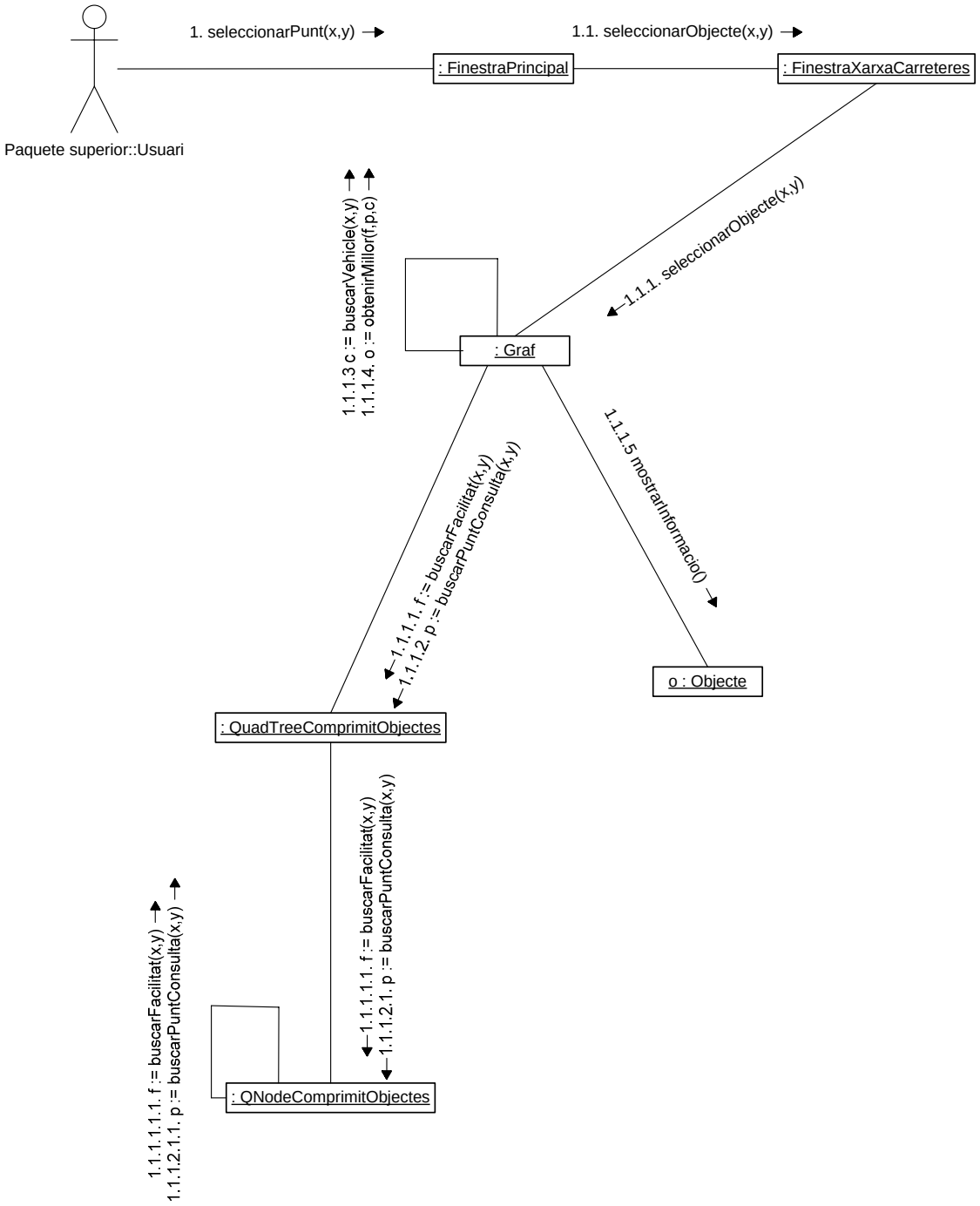


Figura 4.41: Diagrama de col·laboració: Seleccionar objecte

4.3.2.2. Cas d'ús: Marcar Zona

El diagrama de col·laboració següent ens mostra els processos que cal seguir per crear una subxarxa a partir d'un rectangle introduït per l'usuari.

Així doncs, el primer que cal per a marcar una zona és que l'usuari introdueixi dos punts sobre la pantalla per tal de poder crear el rectangle que formarà els límits de la zona.

Llavors, utilitzant el QuadTree de vèrtexs, s'obtindran tots els vèrtexs que formen part d'aquest rectangle i se'n farà una còpia. A totes aquestes còpies se'ls hi assignaran les arestes que també pertanyin a la zona i gràcies a ells, es podran obtenir tots els objectes.

Un cop obtinguts tots els objectes que estiguin dins el rectangle, es crearà un nou graf anomenat graf de la zona. Tot seguit es tractarà aquest graf igual com es fa quan carreguem el fitxer, és a dir, es comprovarà que sigui connex, es crearan les seves estructures de dades i es calcularà la seva parametrització. Finalment, com es pot observar en el pas 2.1.3 del diagrama, s'assignarà el graf de la zona com a l'actual.

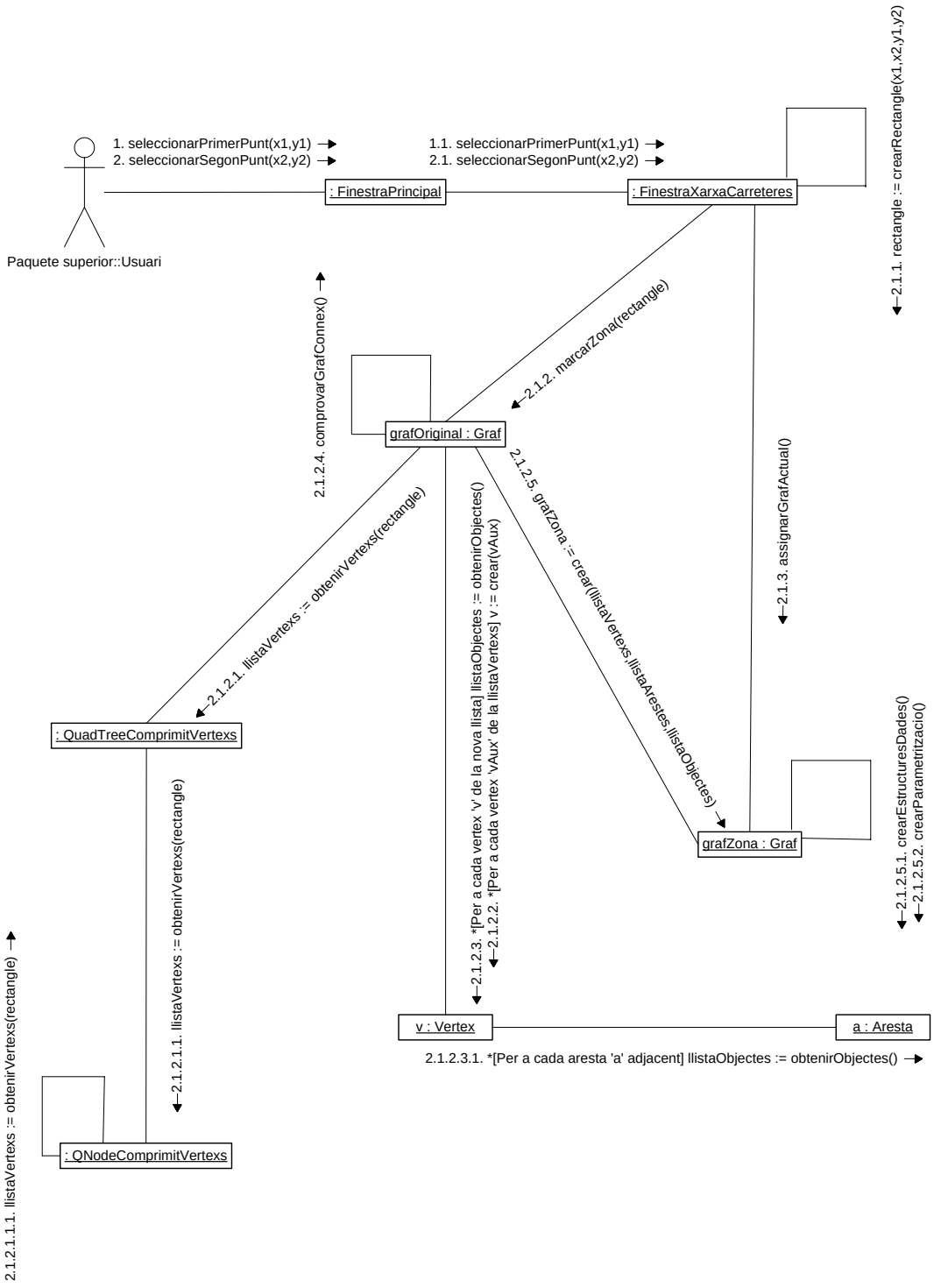


Figura 4.42: Diagrama de col·laboració: Marcar zona

5. Disseny

Un cop arribat a aquest punt, disposem de l'especificació del programari en termes orientats a objectes de manera independent de la tecnologia que s'utilitzarà per implementar-lo. En l'apartat d'anàlisi ens hem centrat en la descripció de cadascuna de les funcionalitats de l'aplicació en abstracte. En aquesta etapa de *disseny del sistema* s'intenta adaptar la documentació generada en el capítol anterior en vistes a la implementació final del sistema mitjançant diverses eines de programació orientada a objectes.

L'aplicació que es vol dissenyar no necessita cap mena de manteniment, és a dir, no existeix cap actor dedicat exclusivament a inserir nous productes, modificar els ja existents, ... Per aquest motiu el sistema a implementar no és considerat com a aplicació de gestió, ja que no cal que el programa disposi d'una part amb accés restringit a usuaris de manteniment.

En aquest capítol no cal realitzar el disseny de l'arquitectura, ni revisar els diagrames descrits en el capítol d'anàlisi ja que tan sols suposaria copiar-los. L'anàlisi del sistema realitzat anteriorment ja s'ha fet pensant força amb el disseny que caldria realitzar en una etapa posterior.

5.1. Disseny de la interfície gràfica d'usuari

En aquest apartat es començarà a parlar sobre la tecnologia utilitzada per implementar la interfície. Aquesta informació serà determinada per la implementació de l'aplicació i, per aquest motiu es pretén definir-la amb antelació per evitar canvis d'última hora. Avançar sobre aquests temes ens facilitarà la posterior etapa ja que ens serà més ràpid i eficaç definir anteriorment totes aquelles tècniques que seran utilitzades en la fase d'implementació de la interfície gràfica d'usuari.

5.1.1. Guia d'estil

Tal i com s'ha comentat, es tractarà de definir cadascuna de les tècniques utilitzades per la construcció de la interfície gràfica d'usuari resultant de l'aplicació.

La tecnologia que s'ha decidit utilitzar per definir la interfície ha estat Qt 4, que ens permet utilitzar tota una sèrie de llibreries de KDE. Es tracta d'una multi plataforma d'eines per interfícies gràfiques d'usuari (GUIs) programades en C++ que ha estat creada i mantinguda per Trolltech. Qt 4 és suportada per un gran nombre de plataformes tot i que la utilitzada en aquesta aplicació ha estat Unix/X11 per Linux.

Les llibreries que Qt 4 aporta permeten al programador construir de manera molt senzilla una sèrie de controls per tal de poder crear una interfície gràfica d'usuari

agradable i molt fàcil d'utilitzar per a qualsevol tipus d'usuari. Cal tenir en compte que la utilització d'aquest tipus d'interfície ens permetrà treballar a través d'una programació orientada a events que ens informaran cada moment del tipus d'interacció que està realitzant l'usuari.

Pel que fa a tot allò que es dibuixa sobre l'espai corresponent a l'escena un cop iniciada l'aplicació (vèrtexs, arestes, objectes, etc.) s'utilitzen comandaments bàsics de les llibreries que aporta OpenGL.

A continuació i com a guia d'estil de l'aplicació que s'està dissenyant es definiran cadascun dels tipus de control utilitzats pel disseny de la interfície gràfica d'usuari. De tot el conjunt d'elements que ens aporta Qt 4 tan sols es descriuran aquells que seran utilitzats. Per a cadascun d'ells s'indicarà la classe que els permet crear així com l'aspecte per defecte que tenen i el tipus d'events associats que han estat utilitzats.

Abans d'indicar un a un els controls utilitzats cal dir que la classe encarregada de construir i de definir l'aspecte d'aquesta interfície gràfica és la classe *FinestraPrincipal*. Dins de la capçalera d'aquesta es fa referència a la classe QObject que representa el punt central del model d'objectes de Qt. Realitzant aquesta instància podrem fer ús dels signals i slots, objectes de comunicació que ens aporten el concepte de programació d'events.

Cal dir també que tot allò que apareix a la finestra (botons, menús, chechbox, ...) s'anomena *widget*. Com es pot observar alguns dels widgets contenen altres widgets a dins. Des del punt de vista del programador es pot pensar que un dels widgets especialitzats hereta d'una classe base anomenada QWidget. Tot seguit es mostren cadascun dels widgets més importants utilitzats en la implementació de la interfície.

5.1.1.1. Widget: QMainWindow

Aquest widget permet construir una finestra principal per a l'aplicació. A partir d'aquesta es podrà afegir un menú i una barra d'eines on s'hi posaran totes les opcions de què es disposa.

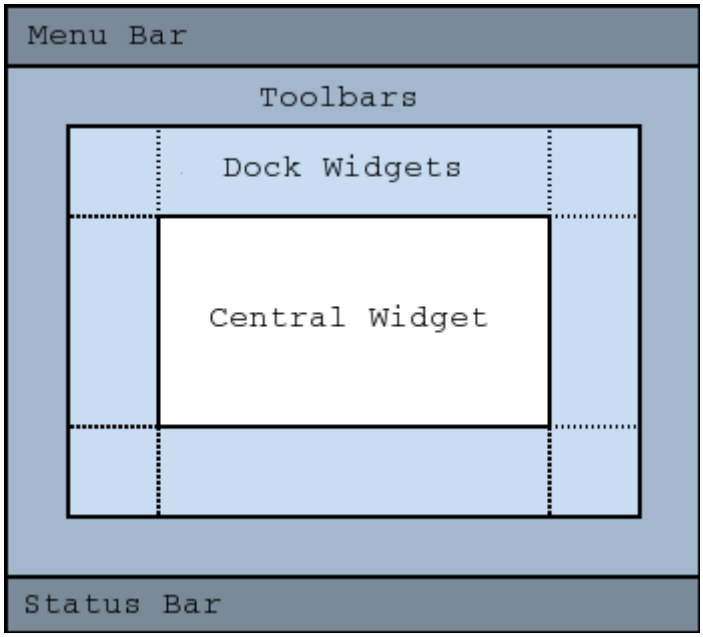


Figura 5.1: QMainWindow

Event: La interacció de l'usuari amb aquests tipus d'objectes es realitzarà a través del menú i la barra d'eines.

5.1.1.2. Widget: QMenuBar

El widget QMenuBar ens permet crear una barra de menú horitzontal clàssica dins la qual hi podrem situar totes aquelles opcions que sigui considerades principals per l'aplicació. Cadascun dels ítems inserits pot disposar d'una llista desplegable que serà mostrada en prémer el botó esquerre del ratolí un cop situats a sobre.

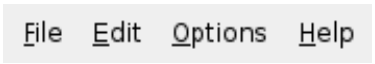


Figura 5.2: QMenuBar

Event: En aquest cas no es produirà cap event interessant per l'aplicació, tot i que internament existeix un que fa que es desplegui el corresponent submenú desplegable que cada ítem tindrà associat.

5.1.1.3. Widget: QMenu

Aquest widget ens permet crear un menú desplegable que pot ser connectat a cadascun dels ítems que conformen el menú principal comentat a l'apartat anterior.

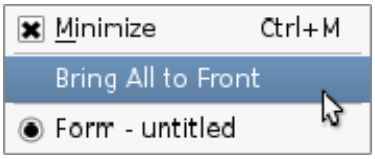


Figura 5.3: QMenu

Event: Cadascun dels ítems d'aquest menú desplegable té associat un event important i utilitzat a la nostra aplicació. Quan algun d'ells és seleccionat s'executa un slot que es troba connectat en tot moment.

5.1.1.4. Widget: QToolBar

Aquest widget permet crear una barra d'eines dins la qual hi podrem situar totes aquelles opcions que es considerin més utilitzades per a l'usuari.

Figura 5.4: QToolBar

Event: Cadascun dels ítems d'aquesta barra d'eines té associat un event important i utilitzat a la nostra aplicació. Quan algun d'ells és seleccionat s'executa un slot que es troba connectat en tot moment.

5.1.1.5. Widget: QWorkSpace

El widget QWorkSpace representa un escriptori on s'hi posaran totes les finestres que es vagin creant. Aquest ens permetrà realitzar canvis de finestra d'una manera més còmode, així com poder visualitzar-ne vàries alhora.

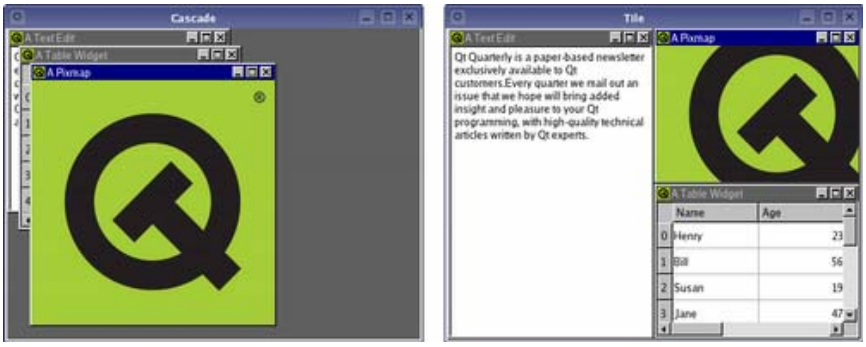


Figura 5.5: QWorkSpace

Event: Aquest widget genera events cada vegada que s'activa una nova finestra. D'aquesta manera es podrà controlar en tot moment quina és la finestra que està utilitzant l'usuari.

5.1.1.6. Widget: QWidget

Aquest widget, anomenat QWidget, és el que permet tenir una finestra flotant per l'escriptori. Aquesta finestra podrà contenir una xarxa de carreteres o bé la seva parametrització.

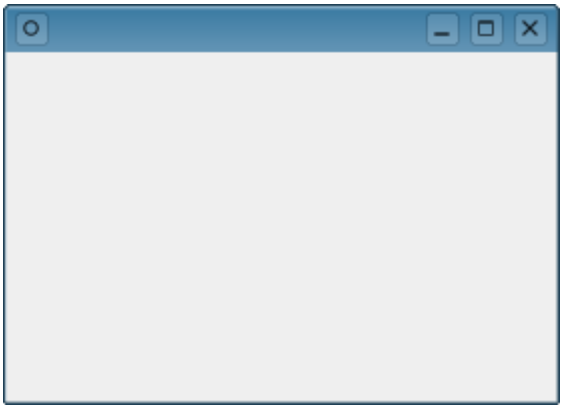


Figura 5.6: QWidget

Event: La interacció de l'usuari amb aquests tipus d'objectes no generarà cap tipus d'event interessant per la nostra aplicació.

5.1.1.7. Widget: QGLWidget

A partir d'aquest widget podrem representar qualsevol figura utilitzant les llibreries proporcionades per OpenGL. Aquest ens permet visualitzar a través d'un objecte de les Qt, gràfics creats amb les comandes bàsiques d'OpenGL. Disposa de tres mètodes necessaris que han de ser redefinits per les classes filles. Aquests mètodes són els següents:

- **paintGL():** Permet visualitzar tot allò que es desitgi sobre l'escena.
- **resizeGL():** Determina el tipus de projecció, la matriu de projecció, etc. en el cas que les mides de la finestra variïn.
- **initializeGL:** Defineix les condicions inicials amb les quals s'haurà de visualitzar l'escena.

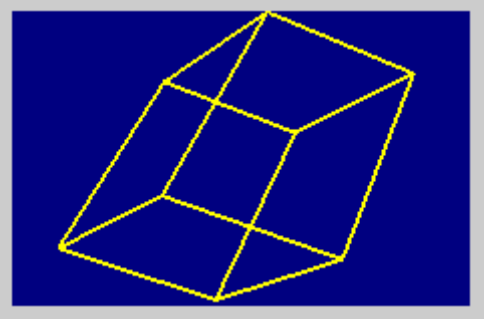


Figura 5.7: QGLWidget

Event: Els events que l'usuari produirà sobre l'escena i que seran detectats seran els següents, tot i que només es tindran en compte en determinades condicions:

- **mousePressEvent(QMouseEvent *m):** Ens avisa quan l'usuari prem qualsevol botó del ratolí en el moment que es trobi sobre l'àrea corresponent de l'escena. A través dels paràmetres podrem saber de quin botó es tracta i sobre quin punt es troba.

- **mouseReleaseEvent(QMouseEvent *m):** Aquest event és executat en el moment en que l'usuari deixa anar el botó del ratolí informant de la mateixa manera que en el cas anterior de quin ha estat el botó que ha deixat de prémer i a quina posició.

- **mouseMoveEvent(QMouseEvent *m):** Ens informa dels moviments que es realitzen amb el ratolí sobre l'escena indicant la posició on es troba en cada moment.

5.1.1.8. Widget: QFileDialog

Es tracta del widget permet obrir i guardar fitxers. Aquesta s'utilitzarà per a carregar un fitxer, per a guardar els objectes i per a guardar el fitxer quan es realitzi una captura de pantalla.

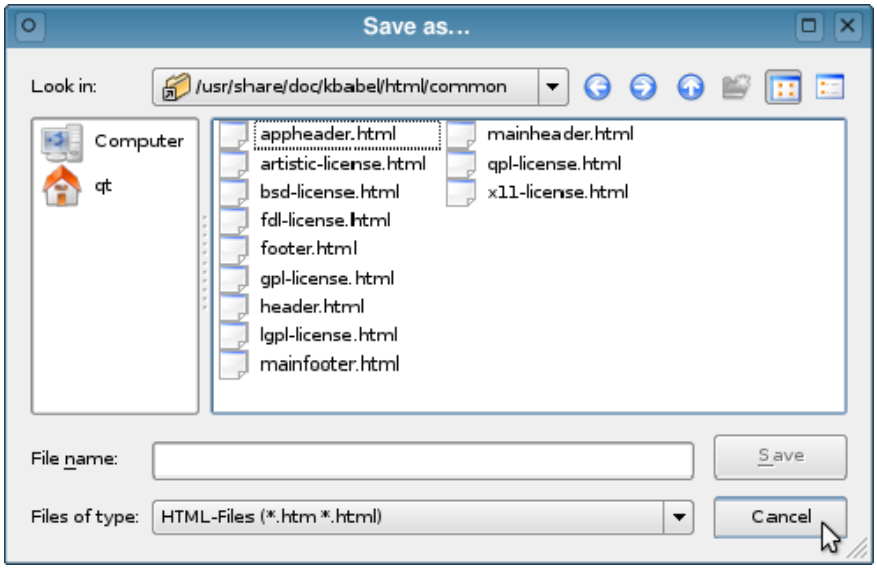


Figura 5.8: QFileDialog

Event: L'event utilitzat és el que ens permet detectar quan l'usuari selecciona un determinat fitxer amb el ratolí, o bé, introduint el nom directament.

5.1.1.9. Widget: QScrollArea

El widget QScrollArea s'utilitza per a poder mostrar un widget que tingui unes dimensions més grans a les que es necessita. Com podem veure a la *figura 5.9*, en el cas que la imatge sigui més gran que la finestra, apareixen automàticament barres d'scroll per a poder veure-la tota.

A l'aplicació, aquest widget s'utilitza quan es vol mostrar una llista d'objectes. Depenent del número que contingui la llista, caldrà posar una barra de scroll per a poder-les veure totes.

Figura 5.9: QScrollArea

Event: Aquest widget no rep ni genera cap event.

5.1.1.10. Widget: QProgressDialog

Aquest widget simplement s'utilitza quan s'ha de realitzar una operació de llarga durada. D'aquesta manera l'usuari pot fer-se una idea del temps que haurà d'esperar-se.

Un dels moments que s'utilitza és quan es carrega una xarxa de carreteres des d'un fitxer. El widget mostra en tot moment quina tasca s'està realitzant i el tant per cent completat.

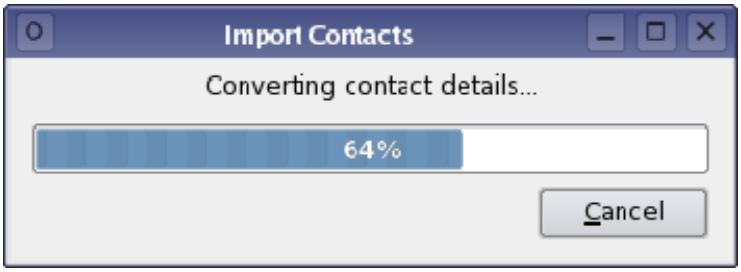


Figura 5.10: QProgressDialog

Event: Aquest widget no té cap event, simplement se li indica el text que ha de mostrar i el tant per cent completat tants cops com es vulgui.

5.1.1.11. Widget: QMessageBox

Aquest widget només s'utilitza per mostrar missatges a l'usuari utilitzant finestres emergents. A més a més, seguit del missatge, l'usuari podrà escollir entre diferents opcions (acceptar, cancel·lar, etc.).

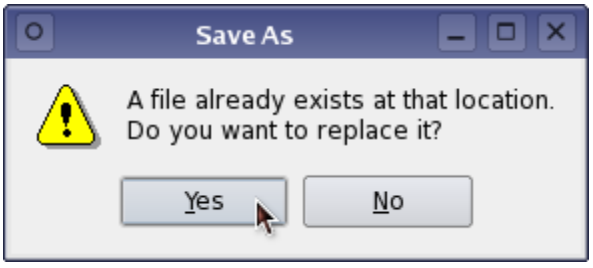


Figura 5.11: QMessageBox

Event: Aquest widget no té cap event, simplement retorna un enter indicant la opció que l'usuari ha escollit.

5.1.1.12. Widget: QVBoxLayout

Aquest widget ens permet construir una mena de taula amb una distribució vertical dins la qual podem ubicar-hi diferents widgets fills. Els diferents atributs estàtics d'aquesta taula podran ser modificats a través dels mètodes dels quals disposa.

Figura 5.12: QVBoxLayout

Event: La interacció de l'usuari amb aquests tipus d'objectes no generarà cap tipus d'event interessant per la nostra aplicació.

5.1.1.13. Widget: QHBoxLayout

Es tracta d'un element igual que l'anterior *(QVBoxLayout)* amb la única diferència del tipus de distribució que en aquest cas és horitzontal.

Figura 5.13: QHBoxLayout

Event: Com en el cas anterior no es tractarà cap event.

5.1.1.14. Widget: QGroupBox

Aquest widget permet posar altres widgets dins d'aquest per tal de crear un grup. Aquesta eina és molt útil per facilitar la visualització dels controls, ja que es posa a cada grup els controls que serveixen per a realitzar una determinada funció.

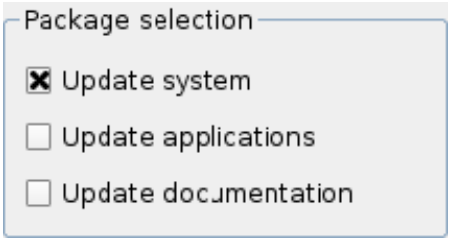


Figura 5.14: QGroupBox

Event: Aquest widget no genera cap event ja que només s'utilitza per agrupar els altres controls.

5.1.1.15. Widget: QButtonGroup

El widget QButtonGroup permet organitzar un conjunt de botons o controls. Organitza una sèrie de comandaments útils per l'aplicació dins de la interfície gràfica.

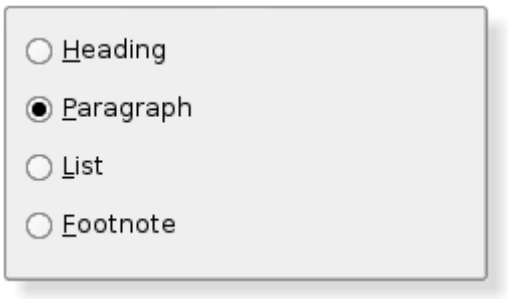


Figura 5.15: QButtonGroup

Event: Els events associats a aquest objecte en si no tenen sentit. Els que són d'interès són els relacionats amb cadascun dels elements interns que conté.

5.1.1.16. Widget: QRadioButton

Un QRadioButton és un objecte format per un indicador rodó seguit d'una etiqueta informativa. L'indicador pot estar seleccionat (punt negre en el seu interior) o no seleccionat (sense cap punt a l'interior). La propietat d'aquests widgets és que, en el cas que hi hagi un conjunt de QRadioButton, només un d'ells podrà estar seleccionat.

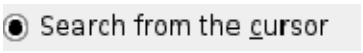


Figura 5.16: QRadioButton

Event: El fet de modificar l'estat dels QRadioButton produeix un event (signal) anomenat *clicked()*. Utilitzant la comanda *connect* podrem relacionar cadascun dels signals amb un slot determinat fent que aquests s'executin quan es produeixi l'event indicat.

5.1.1.17. Widget: QCheckBox

Un QCheckBox és molt semblant a un QRadioButton, amb la única diferència que en el cas que en tinguem un conjunt, podrem tenir-ne seleccionats més d'un. A més, l'indicador té una forma quadrada en comptes de rodona (tal com podem veure a la següent figura.

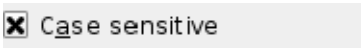


Figura 5.17: QCheckBox

Event: Com en el cas dels QRadioButton, cada QCheckBox pot tenir associada una acció determinada a un event produït. Per tal de produir l'event s'utilitza el signal *clicked()*.

5.1.1.18. Widget: QSpinBox

Els QSpinBox permeten a l'usuari escollir qualsevol valor (un enter) simplement prement els botons del costat dret que permeten incrementar i decrementar el valor. A més, també es permet canviar el número directament dins la caixa tenint en compte que per aplicar aquest nou valor haurem de prémer la tecla *Enter*.

Una de les propietats més important és l'interval de valors que pot prendre. Es poden fixar utilitzant els mètodes *setMinValue(int)* i *setMaxValue(int).*

Figura 5.18: QSpinBox

Event: L'event més habitual en aquest control és *valueChanged(int)* que es dispara cada vegada que el valor és modificat. Aquest passa com a paràmetre el nou valor del widget.

5.1.1.19. Widget: QPushButton

El widget QPushButton permet crear objectes del tipus botó. Aquest tipus de botons donen la sensació d'enfonsar-se en el moment que són premuts. Normalment tenen una acció associada i un cop aquesta ha estat finalitzada el botó torna al seu estat habitual.

Figura 5.19: QPushButton

Event: Existeix un event que permet detectar quan l'usuari prem el botó. Aquest signal s'anomena *clicked()* i pot estar associat com qualsevol event a una acció (slot) que s'executarà quan aquest sigui premut.

5.1.1.20. Widget: QLabel

Un QLabel no és res més que una simple etiqueta que pot estar formada per text planer i per imatges.

Figura 5.20: QLabel

Event: No existeix cap element relacionat íntimament amb una etiqueta dins l'aplicació.

5.1.1.21. Widget: QPixmap

Aquest widget és simplement una imatge que s'utilitza per crear icones. Aquestes icones només s'utilitzen com a enriquiment del menú desplegable.

Figura 5.21: QPixmap

Event: Aquest widget no ens proporcionarà cap event.

5.1.2. Disseny de la interfície

Per tal de dur a terme aquesta aplicació s'ha creat una interfície gràfica d'usuari constituïda per una finestra que conté tots els controls necessaris per la interacció amb l'usuari. Cadascun dels elements que es troben implementats té una funcionalitat associada pel correcte funcionament del sistema pensant sempre en l'objectiu amb el quan ha estat construït.

Per tal de tenir una idea de com serà la interfície gràfica d'usuari es mostra la *figura 5.22* en la qual observarem el resultat final de l'aplicació. D'aquesta manera es podrà anar seguint amb més precisió la descripció de cadascun dels controls que hi apareixen.

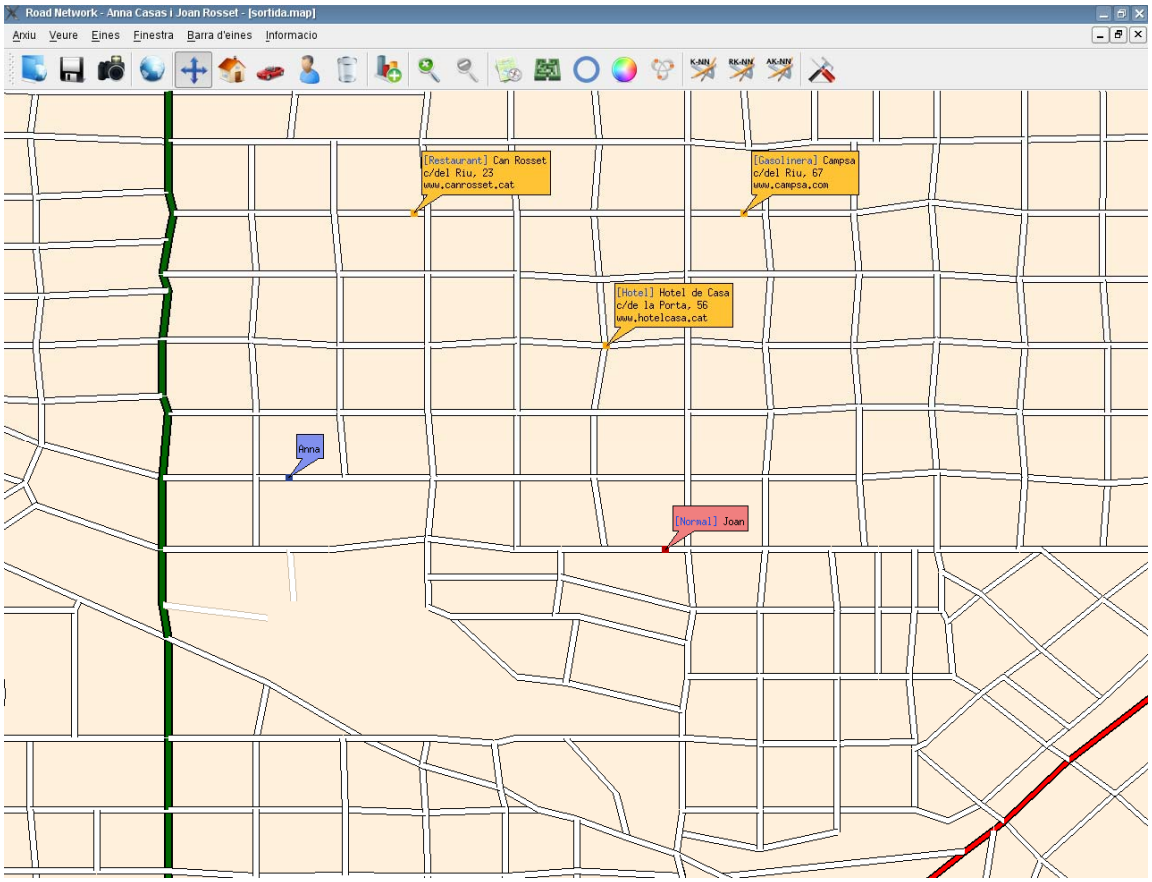


Figura 5.22: Disseny de la interfície gràfica d'usuari

Com podem veure, la interfície conté tots els elements necessaris per poder llegir la xarxa de carreteres, tractar els objectes i resoldre els diferents problemes plantejats.

Aquesta interfície està dividida en tres parts:

- Menú principal.
- Barra d'eines.
- Escriptori.

Per tal d'entendre el funcionament de l'aplicació s'explicaran cadascuna d'aquestes parts:

NOTA: Per possibles problemes amb els accents dins de l'aplicació s'ha optat per no posar-los.

5.1.2.1. Menú Principal

Com hem vist, el menú principal es troba a la part superior de la finestra de l'aplicació. Aquest ha estat creat mitjançant el widget QMenuBar, on cada ítem d'aquest conté un QMenu. A la següent figura podem veure tots els ítems que conté:

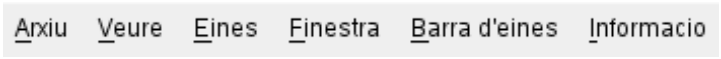


Figura 5.23: Menú principal

Tot seguit s'explicarà amb més detall quins són els ítems dels submenús i les seves funcionalitats.

5.1.2.1.1. Menú: Arxiu

El menú arxiu, tal i com es mostra a la *figura 5.24*, té els següents ítems:

Figura 5.24: Menú: Arxiu

- **Obrir...:** Opció que permet carregar una xarxa de carreteres des d'un fitxer amb l'extensió ".map". S'obre un widget *QFileDialog* on l'usuari haurà d'indicar la ruta del fitxer a carregar.

- **Guardar...:** Permet guardar tots els objectes que hi ha a la xarxa de carreteres en un moment determinat dins un fitxer de text. El nom del fitxer serà el mateix que el nom de la xarxa de carreteres posant-li la extensió ".map_data".

- **Guardar imatge...:** Guarda la imatge de l'escena des del punt de vista i zoom que es troba en aquell moment. Per escollir el nom i la ruta del fitxer a guardar, utilitzem de nou el widget *QFileDialog.*

- **Sortir:** Opció per sortir de l'aplicació.

5.1.2.1.2. Menú: Veure

El menú Veure s'encarrega de mostrar o ocultar els objectes de la xarxa de carreteres. Com es pot veure a la *figura 5.25*, té els següents ítems:

Figura 5.25: Menú: Veure

- **Vèrtexs:** Quan la opció està activada, els vèrtexs del graf es mostren. En cas contrari, els vèrtexs quedaran ocults. Per defecte, aquests no es mostraran ja que la xarxa de carreteres es visualitza millor si només hi ha les arestes (carreteres).

- **Arestes:** Igual que en el cas anterior, aquest ítem indica si s'han de mostrar les arestes o ocultar-los.

- **Facilitats:** Ítem per a mostrar o ocultar les facilitats.

- **PuntsConsulta:** Ítem per a mostrar o ocultar els punts de consulta.

- **Cotxes:** Ítem per a mostrar o ocultar els vehicles.

- **Moviment dels cotxes:** Aquesta opció indica si es vol visualitzar el moviment dels vehicles a través de la carretera. Si es desactiva aquesta opció, els vehicles es visualitzaran aturats, però internament seguiran movent-se. Aquest ítem és molt útil ja que mostrar aquest constant moviment requereix recursos i d'aquesta manera es pot evitar.

- **Engegar/Aturar cotxes:** Aquest ítem és semblant a l'anterior, amb la diferència que en aquest cas els vehicles s'aturen (inclús internament). Amb aquesta opció desactivada els recursos necessaris encara són menors.

- **Vèrtex QuadTree:** Ítem per a mostrar o ocultar el QuadTree de vèrtexs. Aquesta opció serveix per a poder visualitzar sobre la xarxa de carreteres aquesta estructura de dades. L'únic sentit que té aquesta funcionalitat és poder visualitzar com s'ha creat l'estructura de dades.

- **QuadTree d'Objectes:** Ítem per a mostrar o ocultar el QuadTree d'objectes. Igual que en el cas anterior, l'usuari pot comprovar com s'ha creat aquest QuadTree.

- **RTree:** Ítem per a mostrar o ocultar l'RTree d'arestes.

5.1.2.1.3. Menú: Eines

El menú Eines és el que conté la major part de les funcionalitats de l'aplicació. A la *figura 5.26* les podem observar:

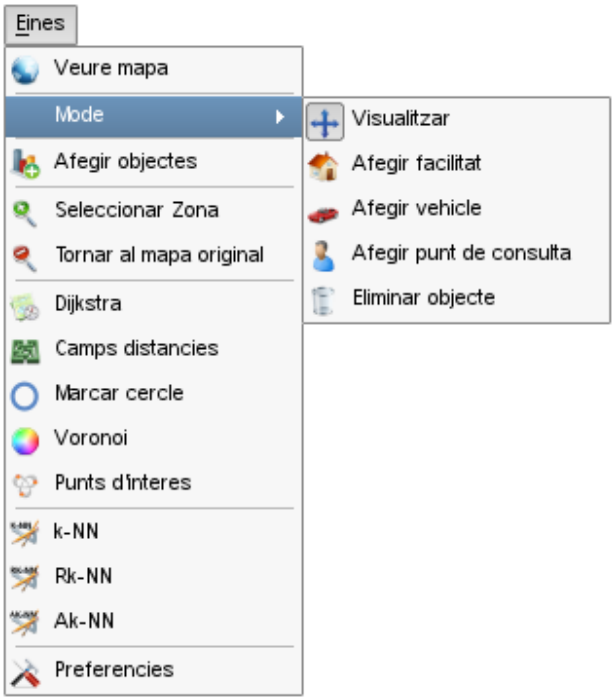


Figura 5.26: Menú: Eines

- **Veure mapa:** Aquesta opció permet a l'usuari visualitzar la xarxa de carreteres i els objectes sense mostrar cap altra cosa. S'utilitza quan l'usuari ha resolt un problema, està veient els resultats per pantalla i necessita tornar a veure la xarxa de carreteres original.

- **Mode:** Aquest ítem conté un submenú on hi apareixen els diferents modes de l'aplicació. Tot seguit s'explica detalladament en què consisteix cadascun d'ells:
  - **Visualitzar:** Aquest mode permet a l'usuari realitzar desplaçaments i zooms a la xarxa de carreteres mitjançant el ratolí.
  - **Afegir facilitat:** El mode afegir facilitat permet a l'usuari seleccionar un punt sobre la xarxa de carreteres i crear-hi una nova facilitat.
  - **Afegir punt de consulta:** Aquest mode, igual que l'anterior, permet a l'usuari afegir un punt de consulta point a la xarxa de carreteres.
  - **Afegir car:** Aquest mode és igual que els dos anteriors, amb la diferència que s'introdueix un vehicle en comptes d'un objecte estàtic.
  - **Eliminar objecte:** Aquest mode permet a l'usuari eliminar un objecte(facilitat, punt de consulta o car) de la xarxa de carreteres.
- **Afegir objectes:** La opció afegir objectes permet a l'usuari afegir tantes facilitats, punts de consulta o vehicles com desitgi. La ubicació d'aquestes nous objectes serà aleatòria.
- **Seleccionar zona:** Aquest ítem permet a l'usuari marcar un rectangle sobre la xarxa de carreteres i crear-ne una de nova utilitzant tots els objectes que hi hagi dins el rectangle.
- **Tornar al mapa original:** Aquesta acció només estarà habilitada quan prèviament l'usuari hagi marcat una zona i necessiti tornar a visualitzar el mapa original.
- **Dijkstra:** La opció Dijkstra permet a l'usuari seleccionar un punt de consulta point inici i una facilitat destí i calcular el camí mínim entre aquests dos punts. Com es pot obviar, l'algorisme utilitzat serà el de Dijkstra.
- **Camps distàncies:** Aquesta opció permet calcular la funció camps distàncies sobre una facilitat de la xarxa.
- **Marcar cercle:** Aquest ítem permet calcular un cercle amb centre a una facilitat i radi introduït per l'usuari.

- **Voronoi:** Aquesta opció permet a l'usuari calcular diagrames de Voronoi d'ordre k.
- **Punts d'interés:** Aquesta acció permet a l'usuari visualitzar quatre punts d'interès sobre la xarxa de carreteres.
- **Problema k-NN:** Aquesta opció permet a l'usuari plantejar un problema de k-veïns més pròxims que l'aplicació ha de resoldre.
- **Problema Reverse k-NN:** Aquesta opció permet a l'usuari plantejar un problema invers de k-veïns més pròxims que l'aplicació ha de resoldre.
- **Problema Aggregate k-NN:** Aquesta opció permet a l'usuari plantejar un problema agregat de k-veïns més pròxims que l'aplicació ha de resoldre.
- **Preferències:** Aquest ítem permet a l'usuari canviar les preferències de l'aplicació.

5.1.2.1.4. Menú: Finestra

El menú Finestra, a diferència dels altres menús, no conté ítems on cadascun d'ells realitza una funció determinada. Aquest menú conté tants ítems com finestres té la nostra aplicació i a través d'aquests es pot activar la finestra desitjada. Si observem la *figura 5.27* podem veure que l'aplicació té dos finestres obertes:

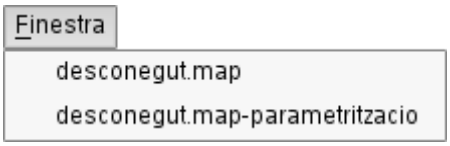


Figura 5.27: Menú: Finestra

5.1.2.1.5. Menú: Barra d'eines

El menú Barra d'eines conté totes aquelles accions que es poden realitzar sobre la barra d'eines de l'aplicació. Tal i com es mostra a la *figura 5.28*, té els següents ítems:

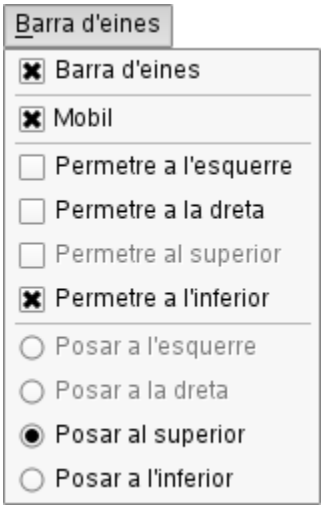


Figura 5.28: Menú: Barra d'eines

- **Barra d'eines:** Ítem per a mostrar o ocultar la barra d'eines.
- **Mòbil:** Ítem que permet a l'usuari moure la barra d'eines o bé deixar-la permanentment en una ubicació determinada.
- **Permetre a esquerra, dreta, superior i inferior:** Aquests quatre ítems permeten a l'usuari indicar a quins dels quatre cantons de l'aplicació pot estar la barra d'eines. Cal remarcar que es pot escollir més d'una opció alhora.
- **Posar a esquerra, dreta, superior o inferior:** Aquests quatre ítems permeten a l'usuari indicar a quina posició vol ubicar la barra d'eines. Com és lògic, només podrà escollir una opció.

5.1.2.1.6. Menú: Informació

El menú Ajuda, tal i com es mostra a la *figura 5.29*, té el següent ítem:

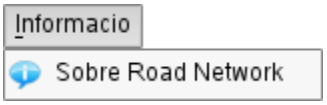


Figura 5.29: Menú: Informació

- **Sobre Xarxa de Carreteres:** Aquesta opció el que fa és obrir una finestra mostrant a l'usuari informació d'interès sobre l'aplicació.

5.1.2.2. Barra d'eines

La barra d'eines és una altra part de la interfície de l'aplicació. Aquesta es troba situada sota el menú principal i conté un conjunt d'ítems que permeten a l'usuari accedir a les opcions més importants. A la següent figura podem veure tots els elements que formen part de la barra d'eines:

Figura 5.30: Barra d'eines

A totes les accions que realitzen els ítems de la barra d'eines també s'hi pot accedir utilitzant el menú principal. Per això, totes aquestes opcions ja estan explicades a l'apartat anterior. Tot seguit s'explicarà a quina acció correspon cada icona de la barra:

Icona	Acció	Ubicació al menú principal
	Carregar fitxer	Menú arxiu/Obrir...
	Guardar objectes	Menú arxiu/Guardar...
	Capturar pantalla	Menú arxiu/Guardar imatge...
	Veure xarxa carreteres	Menú eines/Veure mapa
	Mode visualitzar	Menú eines/Modes/Visualitzar
	Mode afegir facilitat	Menú eines/Modes/Afegir facilitat
	Mode afegir car	Menú eines/Modes/Afegir car
	Mode afegir punt de consulta	Menú eines/Modes/Afegir punt de consulta
	Mode eliminar objecte	Menú eines/Modes/Eliminar objecte
	Afegir objectes	Menú eines/Afegir objectes
	Seleccionar zona	Menú eines/Seleccionar zona

	Desseleccionar zona	Menú eines/Tornar al mapa original
	Camins mínims Dijkstra	Menú eines/Dijkstra
	Funció camps distàncies	Menú eines/Camps distàncies
	Calcular cercle	Menú eines/Marcar cercle
	Calcular Voronoi	Menú eines/Voronoi
	Calcular punts d'interès	Menú eines/Punts d'interès
	Resoldre problema k-NN	Menú eines/K-NN
	Resoldre problema invers k-NN	Menú eines/RK-NN
	Resoldre problema agregat k-NN	Menú eines/AK-NN
	Preferències de l'aplicació	Menú eines/Preferències

Figura 5.31: Barra d'eines: Icones

5.1.2.3. Escriptori

Aquesta part de la interfície és la més gran i és la que s'encarrega de mostrar cadascuna de les finestres. Com hem vist als anteriors capítols, existeixen dos tipus de finestres que s'expliquen a continuació:

5.1.2.3.1. Finestra Xarxa Carreteres

En aquest tipus de finestra es mostra la xarxa de carreteres (vèrtexs i arestes) amb tots els seus objectes (facilitats, punts de consulta, vehicles). L'usuari habitualment utilitzarà aquesta finestra ja que totes les accions dels menús es realitzen sobre una xarxa de carreteres. A la següent figura podem veure un exemple d'aquesta finestra:

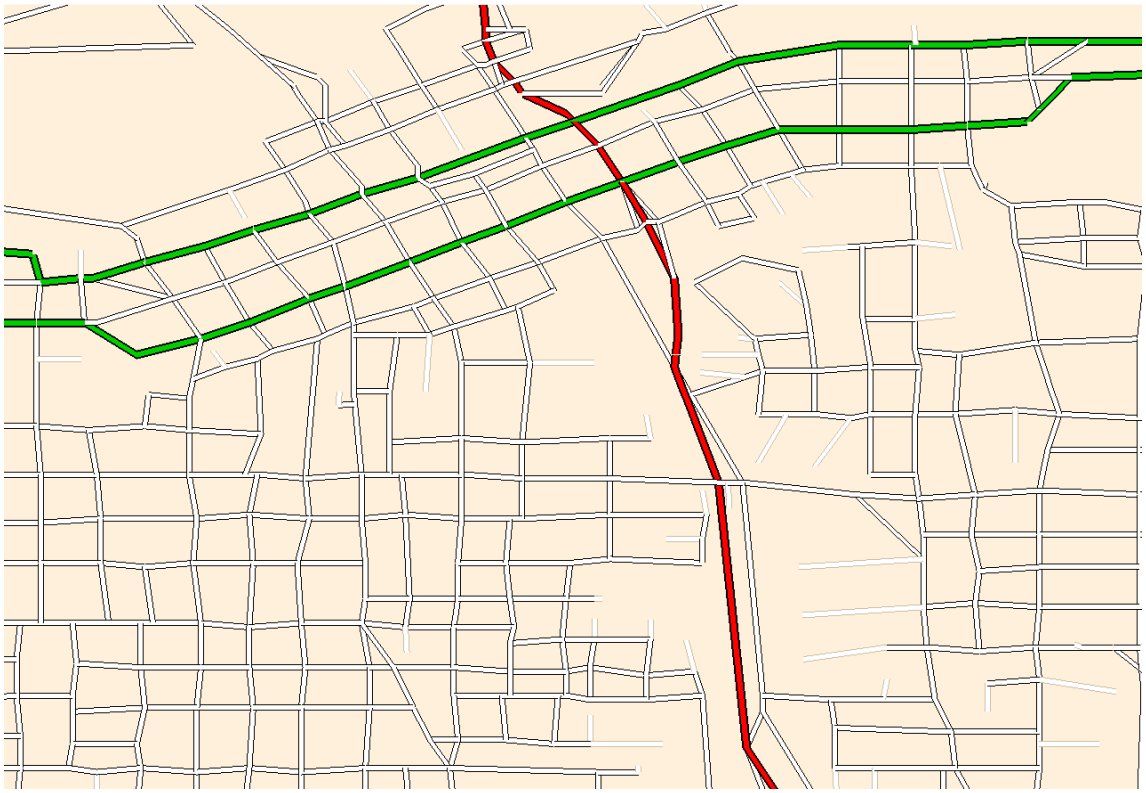


Figura 5.32: Escriptori: Finestra Xarxa Carreteres

5.1.2.3.2. Finestra Parametrització

Aquest tipus de finestra es visualitza la parametrització corresponent a una xarxa de carreteres. Aquesta parametrització és la compactació de la xarxa per tal de poder realitzar els càlculs d'una manera més eficaç i precisa. En aquesta finestra també s'hi pot mostrar el resultat d'algun dels càlculs que es poden realitzar. A la *figura 5.33* podem observar un exemple d'aquest tipus de finestra.

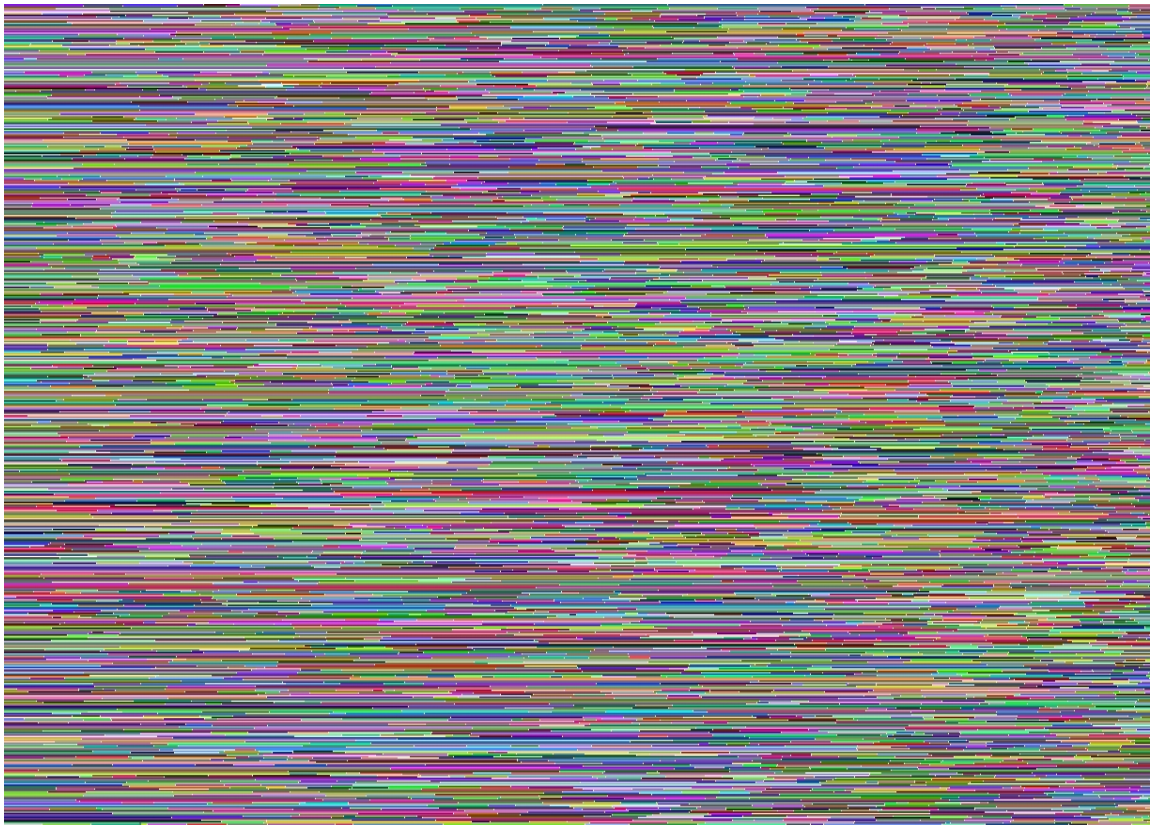


Figura 5.33: Escriptori: Finestra Parametrització

5.2. Disseny de la persistència

Quan un procés finalitza, s'allibera la memòria que aquest utilitzava fent que es perdi tot el seu contingut. En el cas que un objecte creat en un determinat procés hagi de fer-se servir per processos posteriors, cal que aquest es guardi en un dispositiu d'emmagatzematge permanent per tal de poder ser utilitzat. Aquest tipus d'objectes s'anomenen objectes persistents..

En aquest apartat corresponent al disseny de la persistència caldria descriure les bases de dades relacionals que implementen la persistència del sistema tot i que tal i com s'ha comentat a l'inici d'aquest capítol l'aplicació a dissenyar no és de gestió fet que provoca que no existeixi una base de dades relacional.

L'aplicació permet carregar xarxes de carreteres carregar i guardar objectes a partir de fitxers i capturar imatges sobre l'escena. Podem considerar que aquests fitxers representen en certa manera la base relacional de la nostra aplicació, és a dir, el disseny persistent de la mateixa. Per tant, en aquest apartat s'explicarà el format dels fitxers de carreteres i d'objectes i els fitxers d'imatge.

5.2.1. Fitxers de Xarxes

L'aplicació ens permet carregar fitxers de xarxes de carreteres. A partir dels fitxers en format TIGER/Lines s'ha fet un pre-procés i s'ha acabat utilitzant un format més senzill. Els fitxers de xarxes de carreteres no es poden guardar ja que la xarxa en si no pateix modificacions durant la resolució de problemes.

5.2.1.1. Fitxers TIGER/Lines.

La informació geogràfica com carreteres, camins, ferrocarrils, rius, fronteres, etc. és troba representada en els fitxers TIGER/Lines que es creen a partir de la informació geogràfica i cartogràfica extreta del Census TIGER. Aquests fitxers també contenen informació com els noms, tipus i altra informació relacionada.

S'han escollit aquest tipus de fitxers ja que estan molt estesos i són molt complets. En aquest format de fitxer trobem la informació geogràfica dels 50 estats

d'Estats Units, el districte de Columbia, Puerto Rico i les Illes Àreas (Guam, la Mancomunitat Britànica de les Illes Marianas del Nord, Samoa Americana, i les Illes Verges dels Estats Units).

Un fitxer TIGER/Lines està format per varis fitxers que contenen diferent tipus d'informació per a poder generar diversos tipus de mapes o poder obtenir addicional segons les necessitats. Cada un d'aquests fitxers té una extensió concreta. Entre tots aquests fitxers en trobem un amb l'extensió .'RT1' que és el fitxer que conté la informació bàsica de les carreteres.

Per a generar el nostre graf, bàsicament necessitem tenir els vèrtexs, les arestes i el tipus d'aquestes. El fitxer '.RT1', tot i ser el bàsic, conté molta informació que no és necessària per a generar el graf, per tant, s'ha creat un format de fitxer específic per a l'aplicació tan sols amb la informació necessària així permetrà poder llegir més ràpidament.

5.2.1.2. Script: Ruby

Tal i com s'ha explicat en l'apartat anterior necessitem crear un format de fitxer específic per a l'aplicació. El que es farà serà transformar el fitxer '.RT1' del TIGER/Lines en un fitxer més idoni per a la nostra aplicació, el qual tindrà l'extensió '.map'. Per a fer aquesta transformació utilitzarem un script en Ruby.

El format del fitxer resultant, un cop passat l'script serà el següent:

```
0 0 1 2 1 2 A41 A41
1 1 2 3 0 6 3 A41 A41 A41
2 1 0 2 0 4 A41 A41
3 4 2 2 1 5 A41 A41
4 4 0 3 6 2 5 A41 A41 A41
5 5 1 3 3 4 6 A41 A41 A41
6 3.5 0.7 3 1 4 5 A41 A41 A41
```

Figura 5.34: Fitxer xarxa

- Id del vèrtex.
- Longitud de la coordenada.
- Latitud de la coordenada.
- Nombre d'arestes que surten del vèrtex
- Id de cada una de les arestes que surten del vèrtex.
- Tipus de cada una de les arestes que surten del vèrtex.

Aquest fitxer ens dóna tota la informació necessària per a crear la xarxa de carreteres. Podem crear els vèrtexs i les arestes, i a partir d'aquí crear el graf que serà la base de l'aplicació.

5.2.1.3. Connectivitat de la xarxa

En moltes ocasions el graf que obtenim del nostre fitxer de carreteres no és connex, és a dir, no hi ha una connexió entre cada parella de vèrtexs. És habitual que hi hagi petites carreteres que estiguin desconnectades del graf principal.

Degut a la poca importància d'aquestes petites carreteres s'ha decidit suprimir-les per tal de facilitar els càlculs posteriors. Així, s'obté el graf connex que conté el major nombre de vèrtexs.

A la següent figura podem veure la visualització del fitxer de la figura 5.33.

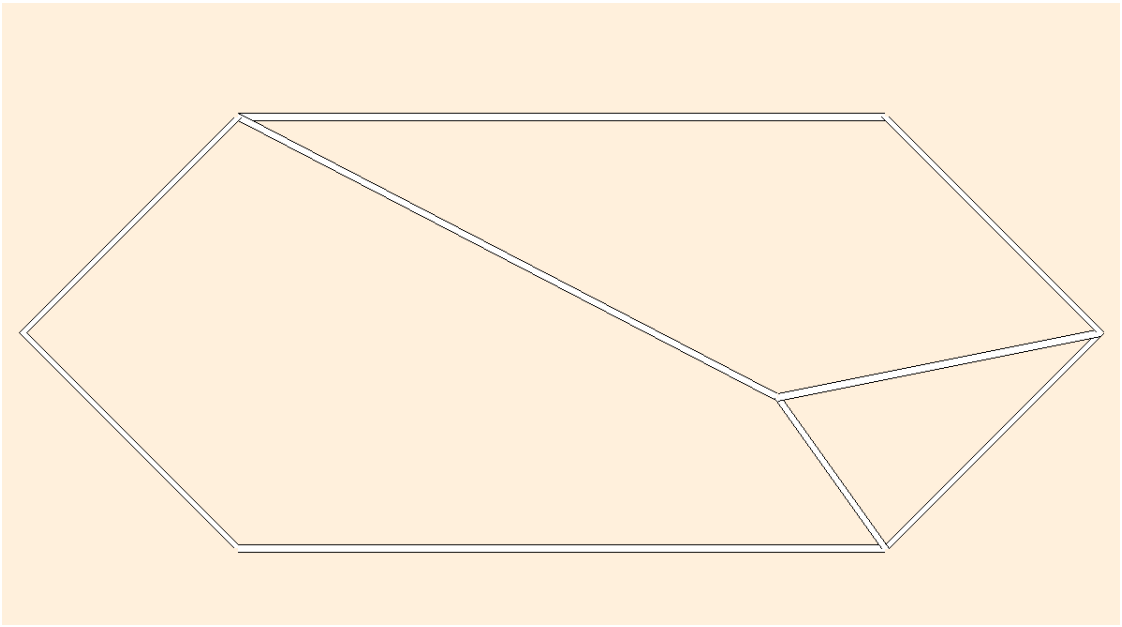


Figura 5.35: Fitxer xarxa: Visualització

5.2.2. Fitxers d'Objectes

Els fitxers d'objectes permeten guardar i carregar objectes creats i tenen l'extensió ".map_data".

Aquests fitxers es poden crear mitjançant l'aplicació o bé manualment. En el cas que es vulgui crear manualment s'haurà d'anar amb molta cura perquè s'haurà de respectar en tot moment el format per no derivar en errors. El programa només podrà carregar objectes des d'un fitxer si aquest té l'extensió ".map_data" i el format correcte.

El fitxers d'objectes pot contenir qualsevol tipus d'objectes, és a dir, facilitats, punts de consulta i/o vehicles. Cada element tindrà el seu format depenent dels atributs necessaris per a crear-lo. Quan es guarden els objectes des de l'aplicació

primer es guarden els objectes, després els vehiclesi finalment els punts de consulta. Cal destacar que per a poder llegir correctament alguns camps com el nom o l'adreça alhora d'escriure'ls al fitxer s'han hagut de substituir els espais que contenien per '_'.

Les facilitats tindran el següent format:

```
  F 2.45661 2 1 3 699 3 0 0 0 Ca_la_Maria Mas_Colom
www.calamaria.com Restaurant
  F 0.5552 0.4448 0 2 565 1 0 0 0 Repsol C/_St._Lluc,_23
www.repsol.com Gasolinera
```

Figura 5.36: Fitxer objectes: Línia facilitat

- 'F' que indica que és una facilitat.
- Longitud de la coordenada.
- Latitud de la coordenada.
- Id vèrtex origen que permet trobar el vèrtex origen.
- Id vèrtex destí que permet trobar el vèrtex destí.
- Valor X de la parametrització de la xarxa.
- Valor Y de la parametrització de la xarxa.
- Component de color R
- Component de color G.
- Component de color B.
- Nom assignat a la facilitat.
- Adreça assignada a la facilitat.
- Pàgina web assignada a la facilitat.
- Tipus de facilitat.

Les punts de consulta tindran el següent format:

```
  Q 4.03386 0.806771 5 6 909 9 0 0 0 Anna_Casas
  Q 0.768985 1.76899 0 1 783 6 0 0 0 Joan_Rosset
```

Figura 5.37: Fitxer objectes: Línia punt de consulta

- 'Q' que indica que és un punt de consulta.
- Longitud de la coordenada.
- Latitud de la coordenada.
- Id vèrtex origen que permet trobar el vèrtex origen.
- Id vèrtex destí que permet trobar el vèrtex destí.
- Valor X de la parametrització de la xarxa.
- Valor Y de la parametrització de la xarxa.
- Component de color R.
- Component de color G.
- Component de color B.
- Nom assignat al punt de consulta.

Els vehicles tindran el següent format:

```
C 1.82193 0 2 4 -1 0 0 0 4 Audi_A3 Normal
```

Figura 5.38: Fitxer objectes: Línia car

- 'C' que indica que és un car.
- Longitud de la coordenada.
- Latitud de la coordenada.
- Id vèrtex origen que permet trobar el vèrtex origen.
- Id vèrtex destí que permet trobar el vèrtex destí.
- Valor X de la parametrització de la xarxa.
- Valor Y de la parametrització de la xarxa.
- Component de color R.
- Component de color G.
- Component de color B.
- Id del vèrtex cap a on es dirigeix el car.
- Nom assignat al car.
- Velocitat que té el car.

A la següent figura podem veure la visualització del fitxer d'objectes format per la unió de les figures 5.36, 5.37 i 5.38. Cada tipus d'objecte es representa a la xarxa d'un color predeterminat. Les facilitats es representen de color taronja, els punts de consulta de color blau i els vehicles de color vermell.

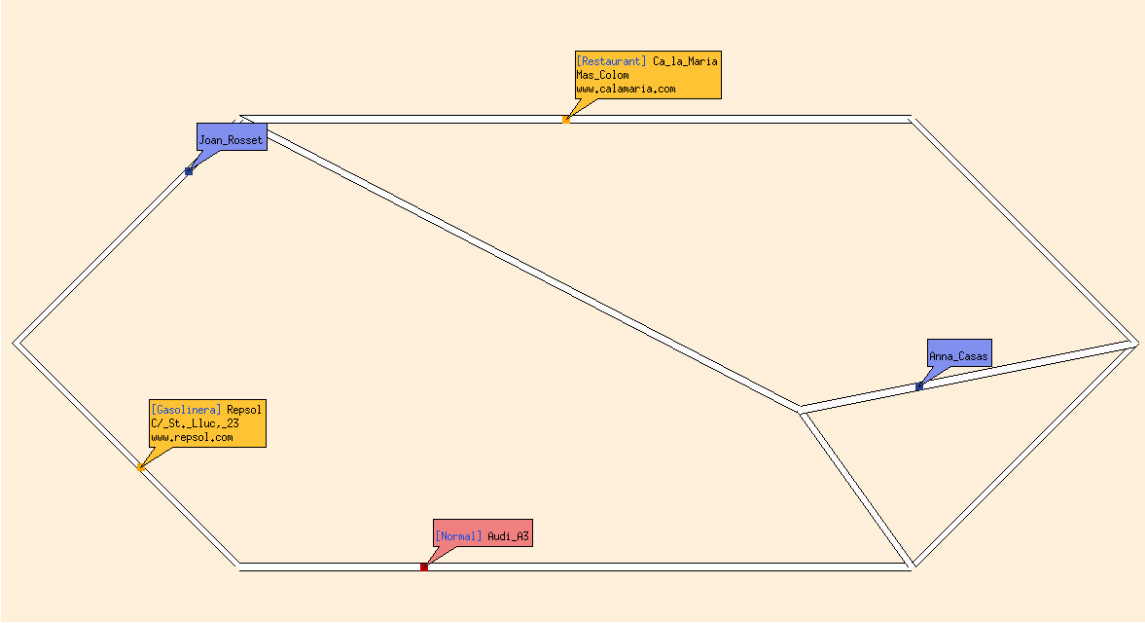


Figura 5.39: Fitxer objectes: Visualització

5.2.3. Fitxers d'imatge

Els fitxers creats per l'aplicació permeten guardar les imatges preses de l'escena en qualsevol moment. Es guardaran en el format ".jpg" o ".bmp" segons determini l'usuari.

6. Implementació

La implementació d'aquesta aplicació s'ha dut a terme utilitzant el llenguatge C++ el qual ens ha aportat diferents conceptes útils per millorar la distribució i eficiència del programa, com són la utilització de classes, l'herència, etc.

Per la implementació de la interfície gràfica s'han utilitzat les llibreries Qt4. També s'han fet servir les llibreries d'OpenGL que permeten representar tot allò que es vulgui sobre la xarxa a través de comandes bàsiques. Finalment, per implementar algunes funcionalitats s'ha necessitat el llenguatge Cg per a poder programar la GPU.

Dins d'aquest capítol, es pretén donar a conèixer els mètodes implementats en l'aplicació final que es consideren més importants pel funcionament d'aquesta seguint un cert ordre coherent. Alguns d'aquests mètodes estan descrits amb un pseudocodi de molt alt nivell per tal de fer-lo fàcilment comprensible i on s'han obviat tots els controls d'errors.

Els conceptes teòrics de les funcionalitats implementades es donaran per suposats ja que estan explicats en el *capítol 2: Estudi previ* d'aquest mateix document.

6.1. QuadTree comprimit de vèrtexs

Com ja s'ha explicat en els capítols anteriors, l'aplicació permet crear una estructura de dades per a ordenar els vèrtexs del graf. Aquesta estructura s'anomena QuadTree comprimit de vèrtexs.

En aquest apartat s'explicarà com ha estat implementada aquesta estructura i quins són els seus mètodes més importants.

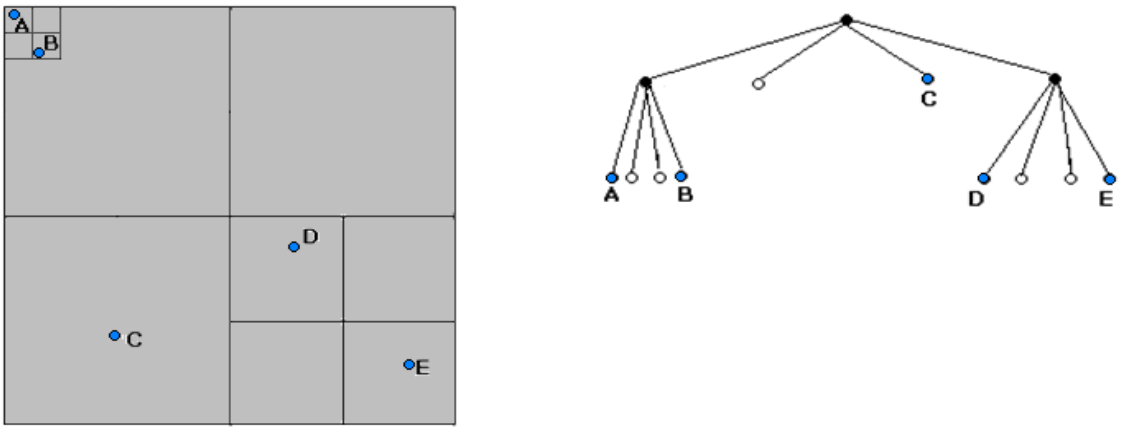


Figura 6.1: QuadTree comprimit de vèrtexs

Primerament es mostrarà com es calcula la caixa englobant del QuadTree comprimit i tot seguit el procés a seguir per a crear-lo. Un cop s'ha creat, aquest permet realitzar consultes d'una manera molt eficient. Dels varis mètodes de consulta

que existeixen, es pot destacar el mètode *seleccionarVèrtexsZona*, el qual permet obtenir tots els vèrtexs que estan dins d'una zona en forma rectangular. Aquest mètode s'explicarà a l'apartat *Seleccionar Zona* d'aquest mateix capítol.

6.1.1. Calcular caixa englobant

La caixa englobant d'un QuadTree comprimit és el quadrat mínim que conté tots els vèrtexs del graf. És a dir, és la zona que ocuparà el node arrel de l'arbre i la qual es començaran a fer particions.

Per a calcular la caixa englobant ens caldrà trobar:

- **Punt d'ancoratge:** És la posició de la cantonada superior-esquerra del quadrat.
- **Mida:** És la mida d'un costat del quadrat.

Els passos a seguir per a trobar aquests dos atributs són:

- Per a trobar la mida s'ha de calcular la màxima distància entre dos vèrtexs dins d'una mateixa coordenada. Per a fer-ho caldrà fer un recorregut per a tots els vèrtexs trobant el valor de la xMínima, xMàxima, yMínima i yMàxima. Amb aquests valors ja es pot calcular la mida del quadrat:

```
mida = màxim(xMàxima-xMínima,yMàxima-yMínima)
```

- Per a trobar el punt d'ancoratge s'agafarà la xMínima i la yMàxima, ja que correspon a la cantonada superior-esquerra:

```
punt d'ancoratge = (xMínima,yMàxima)
```

A la següent figura es pot observar la caixa englobant d'un conjunt de vèrtexs. Podem veure com els dos vèrtexs marcats de color groc sóc els que marquen la distància màxima en una mateixa coordenada (en aquest cas, coordenada x).

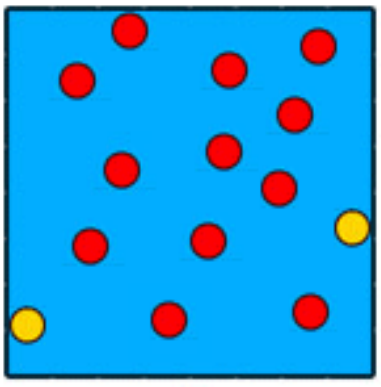


Figura 6.2: QuadTree comprimit de vèrtexs: Calcular caixa englobant

Tot seguit, es mostra aquest mètode en pseudocodi:

```
//Mètode que calcula el punt d'ancoratge i la mida del quadrat que conte tots
els vèrtexs
funció calcularCaixaEnglobant(vector<Vertex> llistaVertexs)

	si llistaVertexs buida llavors
		//No hi ha cap vèrtex: creem caixa de mida 32 amb centre (0,0)
		ancoratge = Coordenada(-16.,16.);
		mida = 32.0;
	altrament
		//Calcular màxima distància entre dos coordenades
		double xMin,yMin,xMax,yMax;

		xMin = llistaVertexs[0]->getLongitud();
		xMax = llistaVertexs[0]->getLongitud();
		yMin = llistaVertexs[0]->getLatitud();
		yMax = llistaVertexs[0]->getLatitud();


		per i de 1 fins llistaVertexs.size()-1 fer
			double x = llistaVertexs[i]->getLongitud();
			double y = llistaVertexs[i]->getLatitud();

			si x<xMin llavors
				xMin=x;
			altrament si x>xMax llavors
				xMax=x;
			fsi
			si y<yMin llavors
				yMin=y;
			altrament si y>yMax llavors
				yMax=y;
			fsi
		fper

		double dist = maxim(xMax-xMin,yMax-yMin);

		si dist == 0.0 llavors
			//Tots els elements tenen el mateix centre o nomes hi ha
			//un element. En aquest cas, el punt on estan tots els
			//vèrtexs serà el centre del quadrat la mida serà 32.
			ancoratge = Coordenada(xMax-16.,yMax+16.);
			mida = 32.;
		altrament
			//Cas normal: agafem el punt d'ancoratge i mida calculats
			ancoratge = Coordenada(xMin,yMax);
			mida = dist;
		fsi
	fsi
ffunció
```

6.1.2. Crear QuadTree comprimit de vèrtexs

Per a crear un QuadTree comprimit a partir d'una llista de vèrtexs, el primer que es fa és calcular la caixa englobant d'aquests vèrtexs. Tot seguit, es crea el node arrel de l'arbre amb el punt d'ancoratge i mida de la caixa calculada.

Un cop es té el node arrel creat es van afegint els vèrtexs, un a un, dins d'aquest node. El mètode utilitzat per a fer aquestes insercions s'anomena *afegirVèrtex.*

6.1.3. Afegir vèrtex

A continuació s'explica com està implementat el mètode *afegirVèrtex*:

- Quan s'afegeix un vèrtex dins el node arrel, el primer que es fa és mirar de quin tipus de node es tracta. Es pot trobar tres tipus diferents de nodes:
  - **Node fulla blanca**: node fulla que no conté cap vèrtex.
  - **Node fulla negra**: node fulla que normalment conté un sol vèrtex. També hi ha la possibilitat que sigui un node amb nivell màxim i aquest pot contenir més d'un vèrtex. Això pot passar perquè, al ser nivell màxim, no pot tenir fills.
  - **Node parcial**: node entremig (conté més d'un vèrtex).

- Depenent del tipus de node que es troba, es realitza la acció corresponent:

  - **Node fulla blanca**: Si s'afegeix un vèrtex en un node fulla blanca, simplement s'insereix a la llista de vèrtexs que conté el node i automàticament es convertirà en una fulla negra.

  - **Node fulla negra**: Si s'afegeix un vèrtex en un node fulla negra, el node tindrà dos vèrtexs i per tant s'haurà de transformar en un node parcial. Per a fer-ho es crearan quatre fills iguals (calculant el punt d'ancoratge i mida de cadascun) i s'afegiran els dos vèrtexs al fill corresponent.
    Per determinar a quin fill s'ha d'afegir un vèrtex s'utilitza el mètode *afegirVèrtexFillCorresponent.* Aquest mètode agafa la coordenada del vèrtex i calcula a quin fill correspon. Tot seguit, afegeix el vèrtex al fill calculat tornant a cridar el mètode *afegirVèrtex* sobre aquest fill.
    En el cas que node fulla negra fos nivell màxim, no podríem crear fills i només caldria afegir el vèrtex a la llista d'aquest node.

  - **Node parcial**: Si s'afegeix un vèrtex en un node parcial, el que es fa és afegir el vèrtex al fill corresponent d'aquest node. El mètode que s'utilitzarà per a afegir el vèrtex al fill que li correspon és el mètode *afegirVèrtexFillCorresponent* esmentat anteriorment.

Cal remarcar que, com que el QuadTree és comprimit, hi ha la possibilitat que s'afegeixi un vèrtex a una zona "morta" i s'hagi d'inserir un nou node entremig.

A la següent figura es pot observar el funcionament del mètode *afegirVèrtex:*

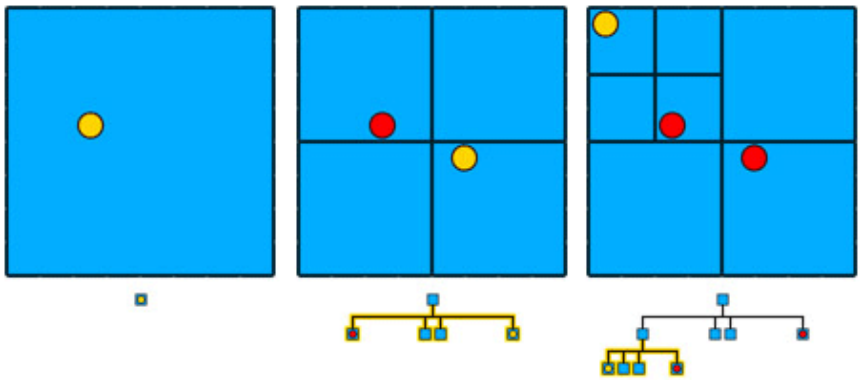


Figura 6.3: QuadTree comprimit de vèrtexs: Afegir vèrtex

- **Imatge 1**: veiem com inicialment hi ha un punt al QuadTree. En aquest moment el QuadTree conté només el node arrel i aquest és fulla negra.

- **Imatge 2**: s'insereix un segon punt i, com que el node arrel és fulla negra, es creen 4 fills iguals i s'insereix cada vèrtex al fill corresponent.

- **Imatge 3**: s'insereix un tercer punt. L'arrel, que és parcial, l'afegeix al seu primer fill. Aquest és fulla negra, per tant, torna a dividir-se en 4 i s'afegeixen els 2 vèrtexs que conté als fills que li corresponen.

Per entendre millor el funcionament dels mètodes *afegirVèrtex* i *afegirVèrtexFillCorresponent* es mostraran en pseudocodi:

```
//Mètode per afegir el vèrtex 'v' a un node
funció afegirVèrtex(Vertex v)
	//Controlem que no dividim masses vegades
	si mida == midaMaxima llavors
		llistaVertexs.afegir(v);
	altrament
		seleccionar tipus
			cas fulla blanca fer
				//Afegim el vèrtex a la llista
				llistaVertexs.afegir (v);
			fcas
			cas fulla negra fer
				//Transformem el node actual en parcial
				//i llavors construïm els fills a través del nou
				//vèrtex i el vèrtex que hi havia a la fulla negra
				llistaVertexs.afegir(v);

				crearFills(); //Creem els fills

				//Afegim els dos fills que tenim als fills creats
				afegirVèrtexFillCorresponent(llistaVertexs[0]);
				afegirVèrtexFillCorresponent(llistaVertexs[1]);
			fcas
```

```
                    cas node parcial fer

                          //Hem de comprovar que el vèrtex formi part d'aquest
                          //node, ja que potser aquest és més petit
                          si this.formaPart(v) llavors
                                //Afegim el vèrtex i l'afegim al seu fill
                                llistaVertexs.afegir(v);
                                afegirVèrtexFillCorresponent(v);
                          altrament
                                //Tenim: el node que ha d'anar el vèrtex és
                                //parcial i, a més, el vèrtex no forma part
                                //d'aquest node
                                //Farem: creem un nou node parcial i el posem
                                //entre el node actual i el seu pare

                                QNodeComprimitVertex aux=crearNodeParcial();
                                aux.calcularAncoratge();
                                aux.calcularMida();

                                //Posem el node aux com a fill del pare del
                                //node actual
                                pare.addNodeFillCorresponent(aux);

                                //Ara hem d'afegir el node actual i el vèrtex
                                //'v' com a fills del node aux.
                                //Hem de controlar que no vagin al mateix
                                //fill, ja que hauríem de transformar el node
                                //'aux' perquè baixés un nivell
                                mentre aQuinFill(this)==aQuinFill(v) fer
                                      //S'ha de recalcular el punt
                                      //d'ancoratge i mida del node aux
                                      aux.baixarNivell();
                                fmentre

                                //Finalment, afegim el node actual i el
                                //vèrtex 'v' al fill aux
                                aux.addNodeFillCorresponent(this);
                                aux.addVèrtexFillCorresponent(v);
                          fsi
                    fcas
              fseleccionar
        fsi
ffunció
```

```
//Mètode que afegeix el vèrtex 'v' en el fill que li correspon
funció afegirVèrtexFillCorresponent(Vertex v)

      //Afegim el vèrtex 'v' al node corresponent
      seleccionar aQuinFill(v)
            cas 1 fer
                  fill1.afegirVèrtex(v);
            fcas

            cas 2 fer
                  fill2.afegirVèrtex(v);
            fcas

            cas 3 fer
                  fill3.afegirVèrtex(v);
            fcas

            cas 4 fer
                  fill4.afegirVèrtex(v);
            fcas
      fseleccionar
ffunció
```

6.2. QuadTree comprimit d'objectes

La aplicació també permet crear una estructura de dades per a ordenar els objectes de la xarxa de carreteres. La estructura de dades que ordena aquests objectes s'anomena QuadTree comprimit d'objectes.

Tot i això, aquesta estructura de dades no pot mantenir els 3 tipus d'objectes que hi ha. El problema és que els vehicles s'estan movent en tot moment i és impossible posar-los dins d'un QuadTree comprimit, ja que s'hauria d'estar reconstruint molt sovint i es perdria molt de temps.

Per tant, aquest QuadTree comprimit ordenarà les facilitats i punts de consulta que formen part de la xarxa de carreteres.

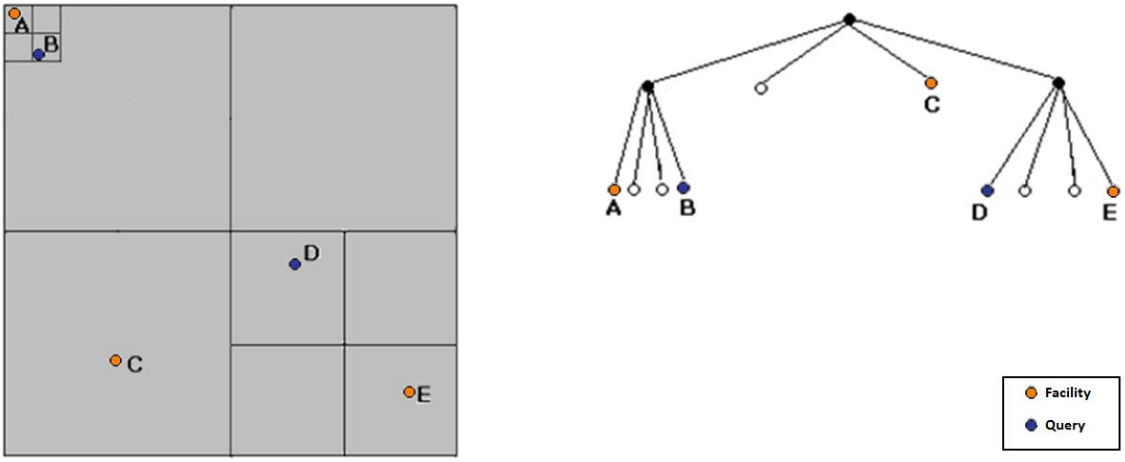


Figura 6.4: QuadTree comprimit d'objectes

Com es pot observar a la *figura 6.4*, el QuadTree comprimit d'objectes tracta igual les facilitats i els punts de consulta i això provoca que sigui molt similar al QuadTree comprimit de vèrtexs.

Així doncs, la única diferència entre el QuadTree comprimit de vèrtexs i el QuadTree comprimit d'objectes és que en comptes de guardar una llista de vèrtexs, caldrà guardar dues llistes: una de facilitats i una altra de punts de consulta.

En aquest apartat s'explicarà com ha estat implementada aquesta estructura i quins són els seus mètodes més importants.

Primerament es mostrarà com es calcula la caixa englobant del QuadTree comprimit i tot seguit el procés a seguir per a crear-lo. Un cop s'ha creat, aquest permet realitzar consultes d'una manera molt eficient. Dels varis mètodes de consulta que existeixen, es pot destacar el mètode *seleccionarObjecte*, el qual permet accedir a un objecte a partir d'una coordenada de pantalla d'una forma molt eficient. Aquest

mètode s'utilitza en gairebé totes les funcionalitats de l'aplicació. La seva implementació s'explicarà a l'apartat *Mantenir objectes* d'aquest mateix capítol.

Cal remarcar que en el QuadTree comprimit de vèrtexs no existeix el mètode *eliminarVèrtex* perquè no és necessari. En canvi en el QuadTree comprimit d'objectes sí que s'ha implementat un mètode anomenat *eliminarObjecte* ja que és una de les funcionalitats de l'aplicació. Aquest mètode també s'explicarà a l'apartat *Mantenir objectes* d'aquest capítol.

6.2.1. Calcular caixa englobant

La caixa englobant d'un QuadTree comprimit és el quadrat mínim que conté tots els objectes(facilitats i punts de consulta) de la xarxa de carreteres. És a dir, és la zona que ocuparà el node arrel de l'arbre i la qual es començaran a fer particions.

Per a calcular la caixa englobant ens caldrà trobar:

- **Punt d'ancoratge:** És la posició de la cantonada superior-esquerra del quadrat.
- **Mida:** És la mida d'un costat del quadrat.

La implementació d'aquest mètode no s'explicarà ja que és pràcticament igual que la del QuadTree comprimit de vèrtexs (explicat a l'apartat anterior) amb la única modificació que s'ha de canviar la llista de vèrtexs per la llista d'objectes.

6.2.2. Crear QuadTree comprimit d'objectes

Per a crear un QuadTree comprimit a partir d'una llista de facilitats i una llista de punts de consulta, el primer que es fa és calcular la caixa englobant d'aquests objectes. Tot seguit, es crea el node arrel de l'arbre amb el punt d'ancoratge i mida de la caixa calculada.

Un cop es té el node arrel creat es van afegint les facilitats i posteriorment els punts de consulta, un a un, dins d'aquest node. El mètodes utilitzats per a fer aquestes insercions s'anomenen *afegirFacilitat* i *afegirPuntConsulta.*

Aquests dos mètodes tampoc s'explicarà com han estat implementats ja que tenen la mateixa estructura que l'*afegirVèrtex* explicat a l'apartat anterior.

6.3. RTree

Com ja s'ha explicat en els capítols anteriors, l'aplicació permet crear una estructura de dades per a ordenar les arestes (arestes) del graf. Aquesta estructura s'anomena RTree.

En aquest apartat s'explicarà com ha estat implementada aquesta estructura i quins són els seus mètodes més importants.

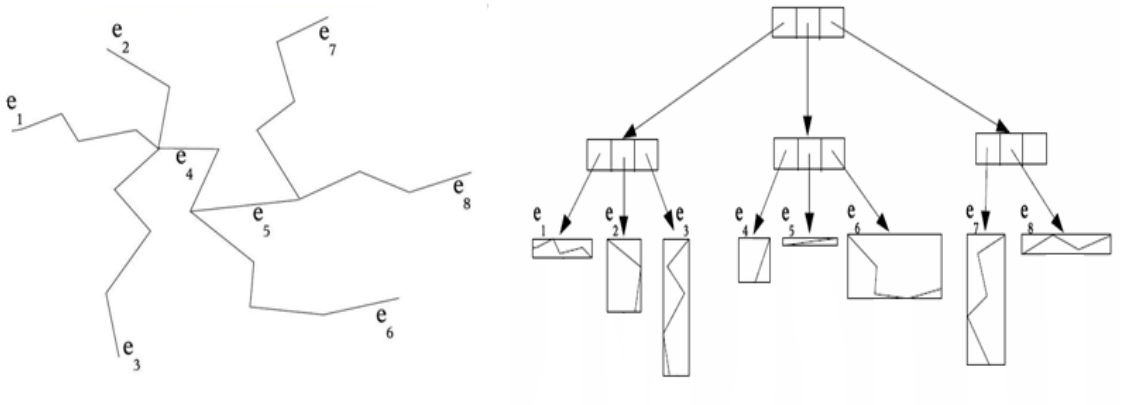


Figura 6.5: RTree

Primerament es mostrarà com es calcula la caixa englobant de forma rectangular de l'Rtree i tot seguit el procés a seguir per a crear-lo. Un cop s'ha creat, aquest permet accedir a les arestes d'una forma molt eficient. Dels varis mètodes de consulta que existeixen, es pot destacar el mètode *seleccionarAresta*, el qual permet obtenir una aresta a partir d'una coordenada de pantalla. Aquest mètode s'utilitza quan l'usuari vol introduir un objecte en una aresta i el selecciona. La seva implementació s'explicarà a l'apartat *Mantenir* objectes d'aquest mateix capítol.

6.3.1. Calcular caixa englobant

La caixa englobant d'un RTree és el rectangle mínim que conté totes les arestes del graf. És a dir, és la zona que ocuparà el node arrel de l'arbre i la qual es començaran a fer particions.

Per a calcular el rectangle englobant ens caldrà trobar:

- **xMin:** És el valor mínim de la coordenada x (longitud) de totes les arestes.
- **xMax:** És el valor màxim de la coordenada x (longitud) de totes les arestes.
- **yMin:** És el valor mínim de la coordenada y (latitud) de totes les arestes.
- **yMax:** És el valor màxim de la coordenada y (latitud) de totes les arestes.

Per a trobar aquests valors caldrà realitzar un recorregut per a totes les arestes del graf. El pseudocodi no es mostra perquè és similar a calcular la caixa englobant d'un QuadTree.

6.3.2. Crear RTree

Per a crear un RTree a partir d'una llista d'arestes, el primer que es fa és calcular la caixa englobant d'aquests. Tot seguit, es crea el node arrel de l'arbre amb els 4 punts del rectangle calculat.

Un cop es té el node arrel creat es van afegint les arestes, un a un, dins d'aquest node. El mètode utilitzat per a fer aquestes insercions s'anomena *afegirAresta.*

6.3.3. Afegir aresta

A continuació s'explica com està implementat el mètode *afegirAresta*:

- Quan s'afegeix una aresta dins el node arrel, el primer que es fa és mirar de quin tipus de node es tracta. Igual que en els QuadTrees, es pot trobar tres tipus diferents de nodes:
  - **Node fulla blanca**: node fulla que no conté cap aresta.
  - **Node fulla negra**: node fulla que conté entre un mínim d'1 aresta i un màxim predefinit. Aquest màxim normalment és 100 però es pot canviar en qualsevol moment per a intentar millorar l'eficiència de l'arbre.
  - **Node parcial**: node entremig. Aquest tipus de node té com a mínim un fill.
- Depenent del tipus de node que es troba, es realitza la acció corresponent:
  - **Node fulla blanca**: Si s'afegeix una aresta en un node fulla blanca, simplement s'insereix a la llista d'arestes que conté el node i automàticament es convertirà en una fulla negra.
  - **Node fulla negra**: Si s'afegeix una aresta en un node fulla negra s'ha de comprovar si hem arribat al màxim establert. En cas d'haver arribat al màxim, haurem de cridar el mètode *dividirRNode* per a dividir el node.
  - **Node parcial**: Si s'afegeix una aresta en un node parcial, el que es fa és afegir l'aresta recursivament al fill que li toca d'aquest node.

Per a decidir a quin fill hem d'afegir l'aresta, el que es fa és calcular la àrea que ocupa cada fill i la àrea que tindria si s'hi afegeix l'aresta. Llavors s'insereix l'aresta al fill que menys àrea li incrementi aquesta inserció.

Per entendre millor el funcionament del mètode afegirAresta es mostrarà en pseudocodi:

```
//Mètode per afegir una aresta a l'RTree
funció afegirAresta(Aresta e)

	//Actualitzem les variables xMin,xMax,yMin i yMax
	si e.getXMin()<xMin llavors xMin = e.getXMin(); fsi
	si e.getXMax()>xMax llavors xMax = e.getXMax(); fsi
	si e.getYMin()<yMin llavors yMin = e.getYMin(); fsi
	si e.getYMax()>yMax llavors yMax = e.getYMax(); fsi

	//Mirem a quin tipus de node estem
	seleccionar tipus
		cas fulla blanca fer
			//Afegim l'aresta al node transformant-lo automàticament en
			//fulla negra
			llistaArestes.afegir(e);
		fcas

		cas fulla negra fer
			//En aquest cas, primer mirem quants elements tenim a la
			//fulla negra, i en cas de tenir-ne més de MAX_ARESTES,
			//llavors haurem de subdividir aquest node.
			si llistaArestes.size()<MAX_ARESTES llavors
				//Encara hi podem posar arestes
				llistaArestes.afegir(e);
			altrament
				//Ja no hi podem posar cap aresta(hem de subdividir)
				llistaArestes.afegir(e);
				dividirRNode();
			fsi
		fcas

		cas node parcial fer
			//En aquest cas hem de mirar a quin fill hem de posar el
			//nova aresta. Per fer-ho mirarem a quin fill ha d'anar i
			//cridarem el mètode recursivament.
			int numFill = aQuinFill(e); //Mirem a quin fill li toca 'e'
			llistaFills[numFill]->afegirAresta(e);
		fcas
	fseleccionar
ffunció
```

6.3.4. Dividir RNode

El mètode *dividirRNode* s'utilitza quan en un node hi ha el màxim d'arestes permesos. El que fa és crear dos fills i repartir tots les seves arestes als seus fills. Aquests fills no tenen perquè ser iguals i potser que es sobreposin entre ells.

A continuació s'explica com està implementat el mètode *dividirRNode*:

- **Pas 1:** Primerament s'agafen les dos arestes de la llista que estiguin més lluny entre ells.
- **Pas 2:** Es creen dos RNodes amb la mateixa caixa englobant que tenen les dos arestes calculats al pas anterior.
- **Pas 3:** Quan es tenen els dos RNodes creats, s'assignen com a els fills del node que estem dividint.
- **Pas 4:** Finalment, per a cadascuna de les arestes del node que estem dividint, els anem inserint al fill que li correspon.
  Per a decidir a quin fill hem d'afegir l'aresta, el que es fa és calcular la àrea que ocupa cada fill i la àrea que tindria si s'hi afegeix l'aresta. Llavors s'insereix l'aresta al fill que menys àrea li incrementi aquesta inserció.

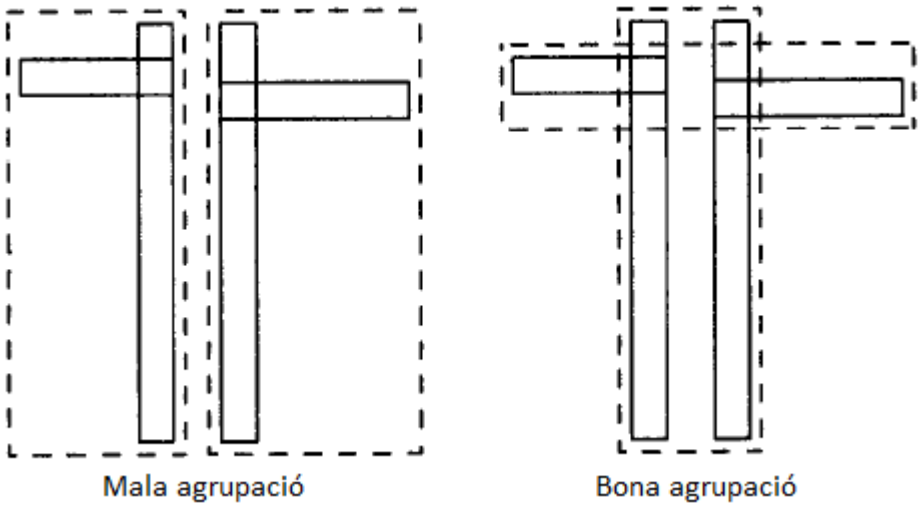


Figura 6.6: RTree: Dividir RNode

A la *figura 6.6* podem observar que per a crear dos grups d'arestes és més important que les regions tinguin àrees mínimes que evitar que es sobreposin.

Per entendre millor el funcionament del mètode *dividirRNode* es mostrarà en pseudocodi:

```
//Operacio per a subdividir el node actual i afegir-hi una nova aresta
funció dividirRNode()

	//Crearem dos nodes on els hi posarem les dues arestes que estiguin mes
	//lluny entre elles
	Aresta arestaGrup1, arestaGrup2;
	double dMax = -1.0;
	per i de 0 fins llistaArestes.size()-1 fer
		per j de i+1 fins llistaArestes.size()-1 fer
			//Calculem d = area(E1,E2) - area(E1) - area(E2)
			// on E1 i E2 son les 2 arestes.
			Aresta aresta1 = llistaArestes[i];
			Aresta aresta2 = llistaArestes[j];

			//Calculem les àrees de les 2 arestes.
			double areaE1 = aresta1.calcularArea();
			double areaE2 = aresta2.calcularArea();

			//Calculem l'àrea dels dos quan formen un rectangle.
			areaTotal = calcularAreaTotal(aresta1,aresta2);

			//Calculem el valor 'd' i guardem les arestes si son els
que			//tenen el valor 'd' més gran.
			d = areaTotal-areaE1-areaE2;

			si d>dMax llavors
				dMax = d;
				arestaGrup1 = aresta1;
				arestaGrup2 = aresta2;
			fsi
		fper
	fper

	//Ja tenim les 2 primeres arestes, ara creem els dos nodes que els
contenen
	RNode nodeGrup1 = new RNode(arestaGrup1,nivell+1);
	RNode nodeGrup2 = new RNode(arestaGrup2,nivell+1);

	//Posem els dos nodes creats com a fills de l'actual
	llistaFills.afegir(nodeGrup1)
	llistaFills.afegir(nodeGrup2);

	//Afegim totes les arestes dins el grup que li correspongui. Per saber
quin	//grup li correspon es calcularà la àrea de cada node i després la àrea
	//de cada node si se li afegís l'aresta. El node escollit serà el que
	//menys àrea li incrementi la inserció de l'aresta.
	per i de 0 fins llistaArestes.size()-1 fer
		double area1 = nodeGrup1.calcularArea();
		double area1Plus = nodeGrup1.calcularArea(llistaArestes[i]);
		double increment1 = area1Plus-area1;

		double area2 = nodeGrup2.calcularArea();
		double area2Plus = nodeGrup2.calcularArea(llistaArestes[i]);
		double increment2 = area2Plus-area2;

		si increment1 < increment2 llavors
			nodeGrup1.afegirAresta(llistaArestes[i]);
		altrament
			nodeGrup2.afegirAresta(llistaArestes[i]);
		fsi
	fper
ffunció
```

6.4. Manteniment d'objectes

Una altra de les funcionalitats de l'aplicació consisteix en mantenir objectes sobre la xarxa de carreteres, és a dir, permetre a l'usuari afegir-ne i eliminar-ne sempre que es vulgui.

Les diferents accions que permet realitzar l'aplicació són:

- Afegir un objecte seleccionant una ubicació i entrant les dades.
- Afegir un conjunt d'objectes amb ubicacions aleatòries.
- Eliminar un objecte seleccionant-la a la pantalla.
- Seleccionar un objecte.
- Moure els vehicles.

Com ja s'ha explicat, per a realitzar aquestes insercions i esborrats s'han creat unes estructures de dades per a millorar-ne l'eficiència.

En aquest apartat s'explicarà com s'han implementat els mètodes més importants que permeten realitzar aquestes tres operacions. Abans, però, s'explicarà una funció utilitzada per a transformar un punt de pantalla en una coordenada de la xarxa de carreteres.

6.4.1. Transformar punt a coordenada

La funció transformar punt de pantalla a coordenada de la xarxa de carreteres permet saber el punt que està seleccionant l'usuari. Aquest mètode s'utilitza molt sovint ja que gairebé totes les funcionalitats de l'aplicació requereix que l'usuari interactuï amb la xarxa de carreteres.

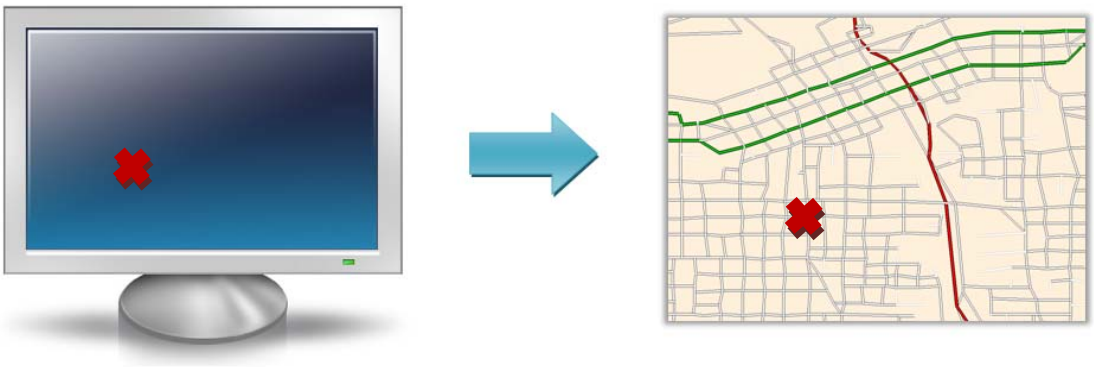


Figura 6.7: Transformar punt a coordenada

Per a realitzar aquesta transformació es fa un càlcul utilitzant el número de píxels de la pantalla i les mides de la xarxa de carreteres.

Per entendre el funcionament del mètode *transformarCoordenades* es mostrarà el seu pseudocodi:

```
funció transformarCoordenades(double xPantalla, double yPantalla)
      //Transformem les variables x i y de pantalla a coordenades de la xarxa
      //de carreteres.
      xXarxa = ( xPantalla * (xMaxXarxa-xMinXarxa) / xNumPixels ) + xMinXarxa;
      yXarxa = ( yPantalla * (yMaxXarxa-yMinXarxa) / yNumPixels ) + yMinXarxa;
ffunció
```

6.4.2. Afegir un objecte

En aquest apartat s'explicarà com ha estat implementat tot el procés que cal seguir per a afegir un objecte a la xarxa de carreteres:

- **Pas 1:** El primer que es fa per a afegir un objecte és seleccionar quin tipus d'objecte es vol afegir: una facilitat, un punt de consulta o un car.

- **Pas 2:** Tot seguit es selecciona un punt a la pantalla indicant la ubicació exacta on es vol inserir. Utilitzant el mètode *transformarCoordenades* (explicat a l'apartat anterior) s'obté la coordenada de la xarxa de carreteres.

- **Pas 3:** Utilitzant l'estructura de dades RTree es calcula si l'usuari ha marcat un punt suficientment a prop d'una aresta. En cas afirmatiu, s'obté l'aresta que ha seleccionat. El mètode de l'RTree que s'encarrega d'aquesta opearció s'anomena *seleccionarAresta* (explicat a continuació).

- **Pas 4:** Quan es té l'aresta seleccionat per l'usuari, s'ha de calcular la coordenada exacta sobre aquesta aresta on s'ubicarà l'objecte. Aquest mètode s'anomena *calcularCoordenadaExacta* (explicat a continuació).

- **Pas 5:** Finalment, es demana que l'usuari entri les dades de l'objecte que ha seleccionat i s'afegeix dins el QuadTree comprimit d'objectes utilitzant el mètode *afegirObjectes* (explicat anteriorment a l'apartat del QuadTree comprimit d'objectes).

A continuació s'explicarà com han estat implementats els mètodes *seleccionarAresta* de l'RTree i el mètode *calcularCoordenadaExacta.*

6.4.2.1. Seleccionar aresta

El mètode *seleccionarAresta* s'utilitza per a obtenir una aresta de l'RTree a partir d'una coordenada de la xarxa de carreteres.

A continuació s'explica com està implementat aquest mètode:

- Quan s'elimina un objecte d'un node, el primer que es fa és mirar de quin tipus de node es tracta. Es poden trobar tres tipus diferents de nodes:
  - **Node fulla blanca**: node fulla que no conté cap aresta.
  - **Node fulla negra**: node fulla que conté entre un mínim d'1 aresta i un màxim predefinit. Aquest màxim normalment és 100 però es pot canviar en qualsevol moment per a intentar millorar l'eficiència de l'arbre.
  - **Node parcial**: node entremig. Aquest tipus de node té com a mínim un fill.
- Depenent del tipus de node que es troba, es realitza la acció corresponent:
  - **Node fulla blanca**: Si s'arriba a una fulla blanca significa que no s'ha trobat cap aresta. En aquest cas es retorna NULL.
  - **Node fulla negra**: Si s'arriba a una fulla negra no significa que s'hagi trobat l'aresta que es buscava. Primer s'haurà de buscar l'aresta més pròxim a la coordenada seleccionada i després es comprovarà si aquesta aresta està suficientment a prop de la coordenada per a seleccionar-lo.
  - **Node parcial**: Si s'arriba a un node parcial haurem de cridar el mateix mètode recursivament. Com ja s'ha explicat, els fills dels RTree poden sobreposar-se entre ells i això fa que possiblement s'hagi de cridar el mètode recursivament a més d'un fill. Si això succeeix, s'agafen totes les arestes resultants i es calcula el que està més a prop de la coordenada.

Per entendre millor el funcionament del mètode *seleccionarAresta* es mostrarà en pseudocodi:

```
//Mètode per buscar una aresta a partir d'una coordenada
funció seleccionarAresta(Coordenada coord) : retorna Aresta

	//Mirem a quin tipus de node estem
	seleccionar tipus
		cas fulla blanca fer
			//S'ha arribat a una fulla blanca, per tant no s'ha trobat
			//cap aresta
			retorna NULL;
		fcas

		cas fulla negra fer
			//S'ha arribat a una fulla negra, es busca l'aresta
			//que estigui més a prop de la coordenada 'coord'
			Aresta aresta = buscarArestaMesProper(coord);

			si aresta.distancia(coord) es gran llavors
				//La distància entre l'aresta trobat i la
				//coordenada 'coord' és gran, per tant
				//es considera que no s'ha trobat cap aresta
				retornar NULL;
			altrament
				//Retornem l'aresta
				retornar aresta;
			fsi
		fcas

		cas node parcial fer
			//Cridem el mètode recursivament

			//Calculem els fills que contenen la coordenada 'coord'.
			//Els fills dels RTree poden sobreposar-se, per això n'hi
			//pot haver més d'un.
			vector<RNode *> fillsPossibles = aQuinsFill(coord);

			//Calculem per a tots els fills possibles l'aresta resultat
			//i retornem el que esta més a prop de la coordenada actual
			Aresta eFinal=NULL, eAux;
			per i de 0 a fillsPossibles.size()-1 fer
				eAux = fillsPossibles[i].seleccionarAresta(coord);
				si eAux és el que està més a prop de coord llavors
					eFinal = eAux;
				fsi
			fper

			retorna eFinal;
		fcas
	fseleccionar
ffunció
```

6.4.2.2. Calcular coordenada exacta

El mètode *calcularCoordenadaExacta* permet trobar la coordenada exacta sobre una aresta a partir d'una coordenada que es trobi a prop d'aquesta aresta.

Com és lògic, l'usuari poques vegades seleccionarà un punt exacte damunt d'una aresta. Aquest mètode el que fa és ajudar-lo a seleccionar una coordenada de forma precisa sobre la xarxa de carreteres.

A la següent figura podem veure un esquema del que pretén resoldre aquest mètode.

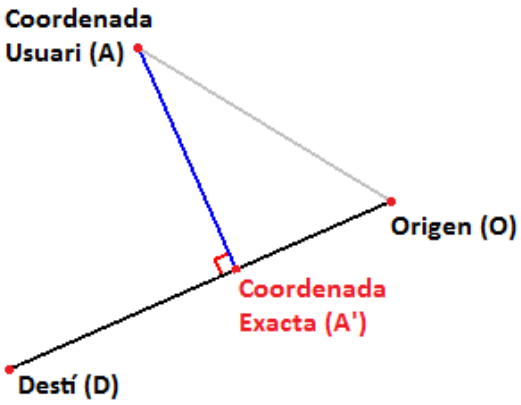


Figura 6.8: Calcular coordenada exacta

Així doncs, a partir de l'aresta OD i la coordenada A, el procés que cal seguir per a trobar la coordenada A' és:

- **Pas 1:** Calcular el vector direcció de l'aresta.

$$\overrightarrow{vectorDirecció} = D - O$$

- **Pas 2:** Calcular el mòdul d'aquest vector.

$$mòdul = \left|\overrightarrow{vectorDirecció}\right| = \sqrt{(vectorDirecció.x)^2 + (vectorDirecció.y)^2}$$

- **Pas 3:** Transformem el vector direcció a unitari.

$$\overrightarrow{vectorDireccióUnitari} = \frac{\overrightarrow{vectorDirecció}}{mòdul}$$

- **Pas 4:** Calcular el vector direcció entre l'origen de l'aresta i la coordenada seleccionada per l'usuari.

$$\overrightarrow{vectorDireccióAuxiliar} = A - V$$

- **Pas 5:** Calcular el producte escalar dels dos vectors OD i OA.

$$prodEscalar = \overrightarrow{vectorDireccióUnitari} \cdot \overrightarrow{vectorDireccióAuxiliar}$$

- **Pas 6:** Calcular A' buscant la coordenada de l'aresta a distància 'prodEscalar' de l'origen.

$$A' = \overrightarrow{Origen} + (\overrightarrow{vectorDireccióUnitari} \cdot prodEscalar)$$

Per entendre millor el funcionament del mètode *seleccionarAresta* es mostrarà en pseudocodi:

```
//Mètode per calcular la coordenada exacta sobre l'aresta més propera a la
coordenada 'coord' passada per paràmetre.
funció calcularCoordenadaExacta(Aresta e, Coordenada coord) : retorna
Coordenada

      //Obtenim l'origen i el destí de l'aresta
      Coordenada coordOrigen = e.getVertexOrigen().getCoordenada();
      Coordenada coordDesti = e.getVertexDesti().getCoordenada();

      //Obtenim el vector direcció de l'aresta
      double vDireccioX = coordDesti.getLongitud()-coordOrigen.getLongitud();
      double vDireccioY = coordDesti.getLatitud()-coordOrigen.getLatitud();

      //Calculem el modul
      double modul = sqrt(vDireccioX*vDireccioX + vDireccioY*vDireccioY);

      //Passem el vector direccio a unitari
      vDireccioX /= modul;
      vDireccioY /= modul;

      //Obtenim el vector entre la coordenada 'c' i l'origen de l'aresta
      double vAuxiliarX = c.getLongitud()-coordOrigen.getLongitud();
      double vAuxiliarY = c.getLatitud()-coordOrigen.getLatitud();

      //Calculem el producte escalar entre vDireccio i vAuxiliar
      double prodEscalar = vDireccioX*vAuxiliarX + vDireccioY*vAuxiliarY;

      //Finalment, agafem la coordenada que esta sobre l'aresta a distancia
      //'prodEscalar' de l'origen
      double xFinal = coordOrigen.getLongitud()+(vDireccioX*prodEscalar);
      double xFinal = coordOrigen.getLatitud()+(vDireccioY*prodEscalar);

      retorna Coordenada(xFinal,yFinal);
ffunció
```

6.4.3. Afegir conjunt d'objectes aleatoris

En aquest apartat s'explicarà com ha estat implementat tot el procés que cal seguir per a afegir un conjunt d'objectes aleatoris a la xarxa de carreteres:

- **Pas 1:** Primerament es demana a l'usuari que esculli quants objectes vol inserir de cada tipus. A més, en el cas de les facilitats se li dóna la possibilitat de seleccionar quin tipus vol que tinguin i en el cas dels vehicles se li deixa seleccionar la velocitat.

- **Pas 2:** Per a afegir cada objecte cal:

  - **Pas 2.1:** Obtenir la llista d'arestes de l'estructura RTree i seleccionar-ne un d'aleatori.

  - **Pas 2.2:** Calcular una coordenada aleatòria sobre l'aresta seleccionat. Per a fer-ho es calcularà el vector direcció de l'aresta i es multiplicarà per a un valor aleatori de 0 a 1.

  - **Pas 2.3:** Calcular el punt de parametrització de la xarxa i crear l'objecte.

  - **Pas 2.4:** Depenent del tipus d'objecte s'afegirà a la xarxa de carreteres amb mètodes diferents:

    - **Facilitat:** s'afegirà al QuadTree d'objectes utilitzant el mètode *afegirFacilitat* (explicat en aquest capítol).

    - **Punt de consulta:** s'afegirà al QuadTree d'objectes utilitzant el mètode *afegirPuntConsulta* (explicat en aquest capítol).

    - **Vehicle:** s'afegirà a una llista de la classe Graf.

6.4.4. Eliminar un objecte

En aquest apartat s'explicarà com ha estat implementat tot el procés que cal seguir per a eliminar un objecte a la xarxa de carreteres:

- **Pas 1:** El primer que cal fer és seleccionar un punt a la pantalla indicant la ubicació de l'objecte a eliminar. Utilitzant el mètode *transformarCoordenades* (explicat en aquest capítol) s'obté la coordenada de la xarxa de carreteres.

- **Pas 2:** Depenent del tipus d'objecte que s'elimini s'utilitzarà un determinat mètode:

  - **Facilitat o Punt de consulta:** Utilitzant l'estructura de dades QuadTree comprimit d'objectes es calcula si l'usuari ha marcat un punt suficientment a prop d'un objecte. En cas afirmatiu, s'elimina l'objecte del QuadTree. El mètode que s'encarrega d'aquesta opearció s'anomena *eliminarObjecte* (explicat a continuació).

  - **Vehicle:** Es fa un recorregut a la llista de vehicles i es busca el que està més a prop de la coordenada seleccionada per l'usuari. En cas que aquest vehicle estigui suficientment a prop de la coordenada, s'elimina.

A continuació s'explicarà com ha estat implementat el mètode *eliminarObjecte* de la classe Quadtree comprimit d'objectes.

6.4.4.1. Eliminar objecte

El mètode *eliminarObjecte* permet eliminar una facilitat o punt de consulta del QuadTree comprimit d'objectes a partir d'una coordenada. Quan s'elimina un objecte el QuadTree ha de quedar correctament modificat.

A continuació s'explica com està implementat aquest mètode:

- Quan s'elimina un objecte d'un node, el primer que es fa és mirar de quin tipus de node es tracta. Es poden trobar tres tipus diferents de nodes:
  - **Node fulla blanca**: node fulla que no conté cap faciliti ni cap punt de consulta.
  - **Node fulla negra**: node fulla que normalment conté una facilitat o un punt de consulta. També hi ha la possibilitat que sigui un node amb nivell màxim i aquest pot contenir més d'un objecte. Això pot passar perquè, al ser nivell màxim, no pot tenir més fills.
  - **Node parcial**: node entremig (conté més d'un objecte).

- Depenent del tipus de node que es troba, es realitza la acció corresponent:

  - **Node fulla blanca**: Si s'arriba a una fulla blanca significa que no s'ha trobat cap objecte a la coordenada que s'ha seleccionat. En aquest cas no cal realitzar res, simplement es retorna fals per a indicar que l'objecte no existeix.

  - **Node fulla negra**: Si s'arriba a una fulla negra no significa que s'hagi trobat l'objecte que es buscava. Primer s'haurà de buscar l'objecte més pròxima a la coordenada seleccionada i després es comprovarà si aquest objecte està suficientment a prop de la coordenada per a eliminar-la.
    Un cop eliminada, el més normal és que la fulla negra s'hagi transformat en fulla blanca. Però en cas de ser nivell màxim podria ser que encara hi quedessin objectes. En aquest cas es comprova si n'ha quedat només una perquè, si fos així, aquest node hauria de canviar el nivell i posar-se només un nivell per sota el seu pare.
    Això succeeix perquè quan es troba més d'un objecte en una fulla negra significa que estem a un node nivell màxim i, per tant, els objectes que conté estan molt a prop. Llavors, quan només queda un objecte dins d'aquest node aquest pot fer-se més gran (fins a ser un nivell per sota el seu pare).

    Nota: Cal recordar que en un QuadTree comprimit potser que un fill tingui varis nivells menys que el seu pare. Si mirem la *figura 6.4*, podem veure com l'arrel, amb nivell 0, té 4 fills. Els fills 2,3 i 4 tenen

nivell 1, ja que la seva mida és la meitat que la del pare. En canvi, si ens fixem amb el fill 1, aquest té nivell 3, ja que la seva mida és 3 vegades més petita que la del pare. Si, per exemple, s'eliminés l'objecte 'A' de la *figura 6.4*, el fill 1 de l'arrel es quedaria només amb un objecte i podria fer-se més gran fins arribar a tenir nivell 1 igual que els seus germans.

- **Node parcial**: Si s'elimina un objecte d'un node parcial, aquest cridarà recursivament la funció *eliminarObjecte* al fill que pertany la coordenada seleccionada. Tot seguit, s'ha de comprovar si el node parcial s'ha quedat amb 3 fills fulles blanques. En aquest cas es posarà el fill que no és fulla blanca en el lloc d'aquest node parcial.

A la següent figura es pot observar el funcionament del mètode *eliminarObjecte:*

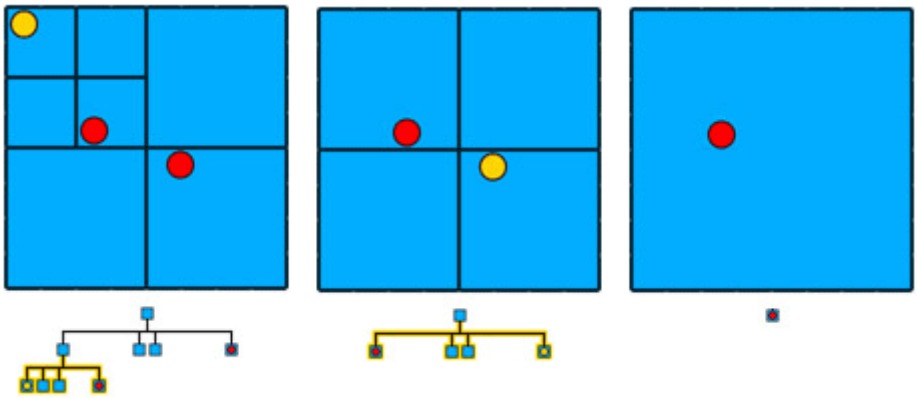


Figura 6.9: QuadTree comprimit d'objectes: Eliminar objecte

- **Imatge 1**: veiem com inicialment tenim un QuadTree amb 3 objectes. En aquest moment decidim eliminar l'objecte marcat de color groc.

- **Imatge 2**: podem observar que després de treure l' objecte de color groc, el fill 1 de l'arrel s'ha quedat amb 3 fills que són fulla blanca. Per tant, l'únic fill que no és fulla blanca passa a ocupar el lloc del seu pare. Ara decidim treure un altre objecte, també marcat de color groc.

- **Imatge 3**: quan s'elimina l' objecte de color groc de la imatge 2, veiem que el node arrel té 3 fulles blanques. Igual que ha passat en el cas anterior, el fill 4 de l'arrel (únic que no és fulla blanca) passa a ocupar el lloc del seu pare (arrel).

Per entendre millor el funcionament del mètode *eliminarObjecte* es mostrarà en pseudocodi:

```
//Mètode per a eliminar un objecte a partir d'una coordenada
funció eliminarObjecte(Coordenada coord) : retorna bool

	seleccionar tipus
		cas fulla blanca fer
			//No s'ha trobat cap objecte en aquesta coordenada
			retornar fals;
		fcas

		cas fulla negra fer
			//Busquem si hi ha un objecte suficient a prop de la
			//coordenada 'coord'. En aquest cas, l'eliminem.
			Objecte loc = buscarObjecteMesProper(coord);

			si loc.distancia(coord) es gran llavors
				//La distància entre l'objecte trobat i la
				//coordenada 'coord' és gran, per tant no la
				//eliminarem.
				retornar fals;
			altrament
				//Eliminem l'objecte
				si loc és una facilitat llavors
					llistaFacilitats.eliminar(loc);
				altrament si loc és un punt de consulta llavors
					llistaPuntsConsulta.eliminar(loc);
				fsi

				//Ara comprovem si el node actual ha quedat com a
				//fulla negra amb un sol objecte. En aquest cas
				//s'ha de modificar el node i fer-lo d'un nivell
				//inferior al pare
				si numObjectes() == 1 llavors
					this.posarNivellInferiorAlPare();
				fsi
				retornar cert;
			fsi
		fcas

		cas node parcial fer
			//Cridem el mètode recursivament
			bool eliminat;
			seleccionar aQuinFill(coord)
				cas 1 fer
					eliminat = fill1.eliminarObjecte(coord);
				fcas
				cas 2 fer
					eliminat = fill2.eliminarObjecte(coord);
				fcas
				cas 3 fer
					eliminat = fill3.eliminarObjecte(coord);
				fcas
				cas 4 fer
					eliminat = fill4.eliminarObjecte(coord);
				fcas
			fseleccionar

			//Un cop s'ha eliminat un objecte d'algun fill s'ha de
			//comprovar si aquest node s'ha quedat amb 3 fills fulles
			//blanques. En aquest cas es posarà el fill que no és fulla
			//blanca al lloc d'aquest node.
			si numFillsFullesBlanques()==1 llavors
				int numFill = numFillNoFullaBlanca();
```

```
                        this.copiarFill(numFill);
                        eliminarFill(numFill);
                    fsi

                    return eliminat;
                fcas
            fseleccionar
ffunció
```

6.4.5. Seleccionar un objecte

El fet de seleccionar un objecte significa buscar-la a partir d'una coordenada de la xarxa de carreteres. Aquest mètode s'utilitza a gran part de les funcionalitats de l'aplicació.

En aquest apartat s'explicarà com ha estat implementat el procés que cal seguir per a obtenir un objecte de la xarxa de carreteres.

- **Pas 1:** El primer que cal fer és seleccionar un punt a la pantalla indicant la ubicació de l'objecte a seleccionar. Utilitzant el mètode *transformarCoordenades* (explicat en aquest capítol) s'obté la coordenada de la xarxa de carreteres.

- **Pas 2:** Depenent del tipus d'objecte que es seleccioni s'utilitzarà un determinat mètode:

    - **Facilitat o Punt de consulta:** Utilitzant l'estructura de dades QuadTree comprimit d'objectes es calcula si l'usuari ha marcat un punt suficientment a prop d'un objecte. En cas afirmatiu, es retorna l'objecte del QuadTree. El mètode que s'encarrega d'aquesta operació s'anomena *seleccionarObjectes* (explicat a continuació).

    - **Vehicle:** Es fa un recorregut a la llista de vehicles i es busca el que està més a prop de la coordenada seleccionada per l'usuari. En cas que aquest vehicle estigui suficientment a prop de la coordenada, es retorna.

A continuació s'explicarà com ha estat implementat el mètode *seleccionarObjectes* de la classe Quadtree comprimit d'objectes.

6.4.5.1. Seleccionar objectes

El mètode *seleccionarObjecte* permet obtenir una facilitat o punt de consulta del QuadTree comprimit d'objectes a partir d'una coordenada. Aquest mètode realitza una cerca sobre el QuadTree i en cap cas el modifica.

A continuació s'explica com està implementat:

- Quan es selecciona un objecte d'un node, el primer que es fa és mirar de quin tipus de node es tracta. Es poden trobar tres tipus diferents de nodes:
  - **Node fulla blanca**: node fulla que no conté cap faciliti ni cap punt de consulta.
  - **Node fulla negra**: node fulla que normalment conté una facilitat o un punt de consulta. També hi ha la possibilitat que sigui un node amb nivell màxim i aquest pot contenir més d'un objecte. Això pot passar perquè, al ser nivell màxim, no pot tenir més fills.
  - **Node parcial**: node entremig (conté més d'un objecte).

- Depenent del tipus de node que es troba, es realitza la acció corresponent:

  - **Node fulla blanca**: Si s'arriba a una fulla blanca significa que no s'ha trobat cap objecte a la coordenada que s'ha seleccionat. En aquest cas es retorna NULL.

  - **Node fulla negra**: Si s'arriba a una fulla negra no significa que s'hagi trobat l'objecte que es buscava. Primer s'haurà de buscar l'objecte més pròxim a la coordenada seleccionada i després es comprovarà si aquest objecte està suficientment a prop de la coordenada. En cas afirmatiu, es retorna l'objecte.

  - **Node parcial**: Si s'arriba a un node parcial, aquest cridarà recursivament la funció *seleccionarObjecte* al fill que pertany la coordenada seleccionada.

Per entendre millor el funcionament del mètode *seleccionarObjecte* es mostrarà en pseudocodi:

```
//Mètode per a seleccionar un objecte a partir d'una coordenada
funció seleccionarObjecte(Coordenada coord) : retorna Objecte

	seleccionar tipus
		cas fulla blanca fer
			//No s'ha trobat cap objecte en aquesta coordenada
			retornar NULL;
		fcas

		cas fulla negra fer
			//Busquem si hi ha un objecte suficient a prop de la
			//coordenada 'coord'. En aquest cas, l'eliminem.
			Objecte loc = buscarObjecteMesProper(coord);

			si loc.distancia(coord) es gran llavors
				//La distància entre l'objecte trobada i la
				//coordenada 'coord' és gran, per tant no la
				//eliminarem.
				retornar NULL;
			altrament
				retornar loc;
			fsi
		fcas

		cas node parcial fer
			//Cridem el mètode recursivament
			Objecte loc;
			seleccionar aQuinFill(coord)
				cas 1 fer
					loc = fill1.eliminarObjecte(coord);
				fcas
				cas 2 fer
					loc = fill2.eliminarObjecte(coord);
				fcas
				cas 3 fer
					loc = fill3.eliminarObjecte(coord);
				fcas
				cas 4 fer
					loc = fill4.eliminarObjecte(coord);
				fcas
			fseleccionar

			retornar loc;
		fcas
	fseleccionar
ffunció
```

6.4.6. Moure els vehicles

Una altra de les funcionalitats de l'aplicació consisteix en moure els vehicles sobre la xarxa de carreteres.

A continuació s'explicarà el procés que cal seguir per a mantenir els vehicles en moviment:

- **Pas 1:** Quan l'usuari decideix engegar els vehicles, es crea un nou fil d'execució paral·lel al fill d'execució principal.
- **Pas 2:** Dins el nou fil d'execució s'entrarà dins un bucle on, per a cada iteració, es mouran tots els cotxes. A cada iteració, aquest fil d'execució s'adormirà 1 segon ja que sinó no es podria veure el moviment.
- **Pas 3:** Quan l'usuari decideix parar els vehicles, automàticament se surt del bucle i s'elimina el nou fil d'execució.

A la següent figura es pot veure gràficament el funcionament d'aquesta funcionalitat.

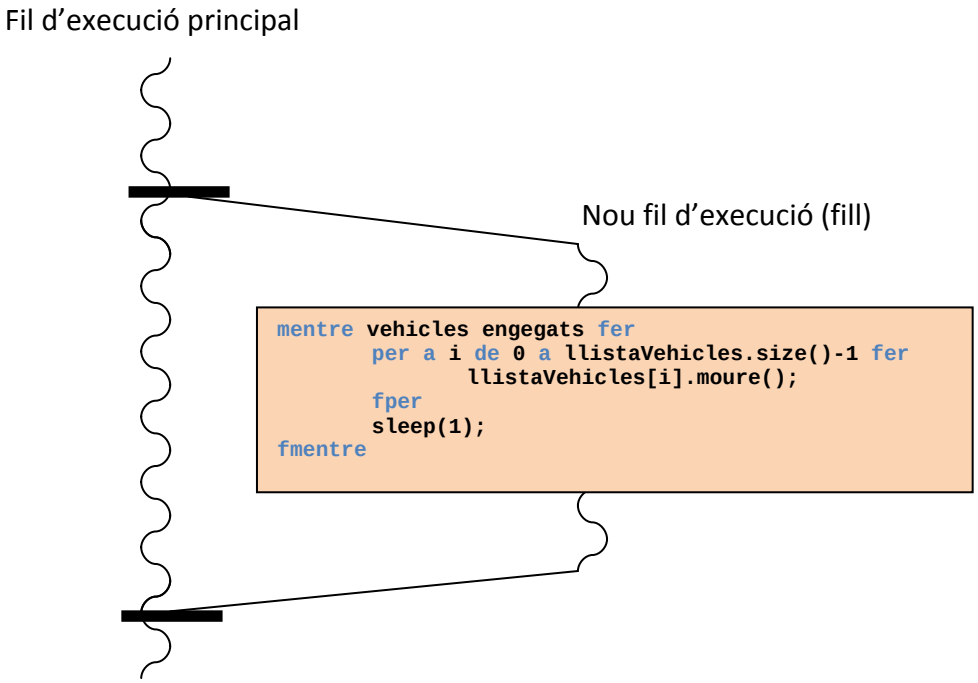


Figura 6.10: Moure vehicles

Com es pot observar, el mètode encarregat de desplaçar els cotxes s'anomena *moure.* Aquest mètode té predefinida una longitud fixa que correspon al moviment que realitzarà el vehicle. Aquesta longitud depèn de la seva velocitat. Cal remarcar que cada vegada que el vehicle es troba en una intersecció, es genera un valor aleatori per a decidir quin serà la següent aresta.

6.5. Seleccionar zona

La funcionalitat seleccionar zona permet crear un subgraf a partir d'una zona rectangular introduïda per l'usuari. Aquesta opció és de gran utilitat quan s'han de resoldre problemes a una part de la xarxa ja que els càlculs es realitzen només en aquesta zona.

En aquest apartat s'explicarà com ha estat implementat tot el procés que cal seguir per a crear una subxarxa de carreteres:

- **Pas 1:** Primerament l'usuari ha de seleccionar dos punts a la pantalla formant un rectangle. Per a crear aquest rectangle serà necessari utilitzar el mètode *transformarCoordenades* (explicat en aquest capítol) per a transformar els punts de pantalla a coordenades de la xarxa.

- **Pas 2:** Utilitzant l'estructura de dades QuadTree comprimit de vèrtexs, es seleccionen tots els vèrtexs que es troben dins la zona i s'afegeixen dins una llista. El mètode del QuadTree comprimit de vèrtexs que s'encarrega d'aquesta operació s'anomena *seleccionarVèrtexsZona* (explicat a continuació).

- **Pas 3:** Realitzar una còpia dels vèrtexs obtinguts i eliminar les seves arestes adjacents que surten de la zona.

- **Pas 4:** Obtenir tots els objectes de les arestes, crear el graf i comprovar la seva connectivitat perquè sigui connex. Si no és connex, eliminar les parts petites no connexes a ell.

- **Pas 5:** Finalment, caldrà calcular la parametrització de la xarxa i les estructures de dades del nou graf, igual com es fa quan es carrega una xarxa de carreteres des d'un fitxer.

A continuació s'explicarà com han estat implementat el mètode *seleccionarVèrtexsZona* del QuadTree comprimit de vèrtexs.

6.5.1. Seleccionar vèrtexs zona

A continuació s'explica com està implementat el mètode *seleccionarVèrtexsZona*:

- Quan s'entra a un node per a obtenir els vèrtexs que estan continguts dins la zona passada per paràmetre, el primer que es fa és calcular si el node està totalment contingut dins la zona. Si és així, es retornen tots els vèrtexs que conté. Per a esbrinar si el node està contingut dins la zona s'utilitza el mètode *contingut* (explicat a continuació).

- En cas que el node no estigui contingut dins la zona, es calcularà si hi interseca per algun costat. El mètode que permet esbrinar-ho s'anomena *intersecat* (explicat a continuació). Si és així, es mirarà a quin tipus de node es troba.
  - **Node fulla blanca**: node fulla que no conté cap vèrtex.
  - **Node fulla negra**: node fulla que normalment conté un sol vèrtex. També hi ha la possibilitat que sigui un node amb nivell màxim i aquest pot contenir més d'un vèrtex. Això pot passar perquè, al ser nivell màxim, no pot tenir fills.
  - **Node parcial**: node entremig (conté més d'un vèrtex).

- Depenent del tipus de node que es troba, es realitza la acció corresponent:

  - **Node fulla blanca**: Si s'arriba a un node fulla blanca, al no tenir cap vèrtex, es retorna una llista buida.

  - **Node fulla negra**: Si s'arriba a una fulla negra, es mira per a cada vèrtex que conté si forma part de la zona i s'afegeix a la llista que es retornarà.

  - **Node parcial**: Si s'arriba a un node parcial, aquest cridarà recursivament el mètode *seleccionarVèrtexsZona* per a tots els seus fills afegint tots els vèrtexs obtinguts pels fills a una llista que, posteriorment, retornarà.

A la següent figura es pot observar el que pretén fer el mètode *seleccionarVèrtexsZona:*

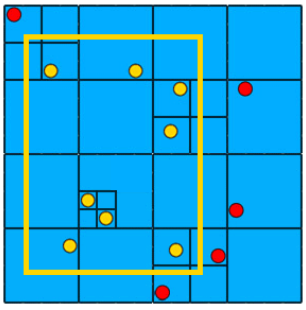


Figura 6.11: Seleccionar vèrtexs zona

Per entendre millor el funcionament dels mètodes *seleccionarVèrtexsZona, contingut i intersecat* es mostraran en pseudocodi:

```
//Mètode que retorna tots els vèrtexs que es troben dins la zona delimitada
//pel rectangle format per les coordenades 'c1' i 'c2'.
funció seleccionarVertexsZona(Coordenada c1, Coordenada c2) : retorna llistaV

	si contingut(c1,c2) llavors
		//El node esta dins la zona, per tant, agafem tots els seus
		//vèrtexs
		retorna llistaVertexs;

	altrament si intersecat(c1,c2) llavors
		//Alguna part del node forma part de la zona, per tant, agafarem
		//només una part dels vèrtexs
		vector<Vertex *> llistaRetorn = vector<Vertex *>();

		seleccionar tipus
			cas fulla blanca fer
				//No cal fer res perquè no hi ha cap vèrtex
				retorna llistaRetorn;
			fcas

			cas fulla negra fer
				//Si es fulla negre, nomes agafarem els que estiguin
				//dins la zona
				per a i de 0 fins llistaVertexs.size()-1 fer
					si llistaVertexs[i].formaPart(c1,c2) llavors
						llistaRetorn.afegir(llistaVertexs[i];
					fsi
				fper
			fcas

			cas node parcial fer
				//Es node parcial, per tant, cridem recursivament el
				//mètode pels seus fills
				llistaRetorn.afegir(fill1.seleccionarVertexsZona(c1,c2));
				llistaRetorn.afegir(fill2.seleccionarVertexsZona(c1,c2));
				llistaRetorn.afegir(fill3.seleccionarVertexsZona(c1,c2));
				llistaRetorn.afegir(fill4.seleccionarVertexsZona(c1,c2));
			fcas
		fseleccionar

		retorna llistaRetorn;
	fsi

	//Si el node està fora de la zona, es retorna una llista buida
	retorna vector<Vertex *>();
ffunció
```

```
//Mètode que retorna cert si el node forma part de la zona, altrament fals
funció contingut(Coordenada c1, Coordenada c2) : retorna bool
	//Mirem si el node esta dins la zona
	si (!(basePoint.getLongitud()>=min(c1.getLongitud(),c2.getLongitud())))
		retorna fals;
	fsi
	si (!(basePoint.getLongitud()+length<=max(c1.getLongitud(),c2.getLongitud())))
		retorna fals;
	fsi
	si (!(basePoint.getLatitud()<=max(c1.getLatitud(),c2.getLatitud())))
		retorna fals;
	fsi
	si (!(basePoint.getLatitud()-length>=min(c1.getLatitud(),c2.getLatitud())))
		retorna fals;
	fsi

	retorna cert;
```

```
ffunció
```

```
//Mètode que retorna cert si alguna part del node forma part de la zona,
//altrament fals
//Precondició: contingut és fals
funció intersecat(Coordenada c1, Coordenada c2) : retorna bool

       //Mirem si el node esta dins la zona
       si (!(basePoint.getLongitud()<=max(c1.getLongitud(),c2.getLongitud())))
              retorna fals;
       fsi
       si (!(basePoint.getLongitud()+length>=min(c1.getLongitud(),c2.getLongitud())))
              retorna fals;
       fsi
       si (!(basePoint.getLatitud()>=min(c1.getLatitud(),c2.getLatitud())))
              retorna fals;
       fsi
       si (!(basePoint.getLatitud()-length<=max(c1.getLatitud(),c2.getLatitud())))
              retorna fals;
       fsi

       retorna cert;
ffunció
```

6.6. Càlcul de la Parametrització de la Xarxa

La parametrització és una manera de compactar la xarxa per a posteriorment poder-hi realitzar càlculs.

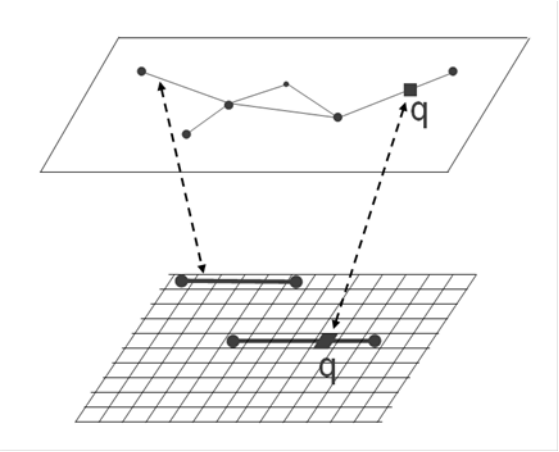


Figura 6.12: Parametrització de la xarxa

Per a poder compactar cadascuna de les arestes de la xarxa de carreteres s'utilitza una reixa rectangular de píxels on s'hi mapegen les arestes, una al costat de l'altra, amb un nombre determinat de píxels. Cada aresta té una longitud determinada en la xarxa però cal determinar quants píxels tindrà en la parametrització. El nombre de píxels haurà de ser proporcional a la seva longitud, així com més llarga sigui una aresta més píxels tindrà en la parametrització. Per tant, el primer que hem de calcular és el nombre de píxels per unitat de longitud que toquen a cada aresta. Per a fer-ho hem de dividir el nombre de píxels de què disposem per la longitud total de les

arestes. El nombre de píxels ve determinat pel width i el height que per defecte correspondran a la mida de la pantalla (n'agafarem menys ja que amb arrodoniments i espais buits a final de línia podem perdre espai).

Un cop fet aquest càlcul i abans de poder començar a mapejar les arestes, creem un multimap[2] (que anirà acompanyat d'iteradors) per controlar quines files de la reixa tenim disponibles i quant d'espai buit hi resta.

Ara, per a cada una de les arestes i mentre quedi espai disponible:

- Es calcula el nombre de píxels que necessita l'aresta.
- Cal trobar una fila per col·locar l'aresta:
  - Si el nombre de píxels és més gran que el width (píxels que té una fila) es col·loca l'aresta en tota una fila buida, si encara en queden, o en la fila que tingui més espai disponible.
  - Altrament es busca una fila amb prou espai.
- Es guarden els punts inicial i final de parametrització a l'aresta.
- Si la fila ha quedat plena cal eliminar-la del multimap, si no ha quedat plena cal actualitzar el nombre de píxels disponibles.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularParametritzacio()

      //calculem la longitud de totes les arestes i el nombre de píxels/unitat
      double longitudTotal=calcularLongitudArestes();
      int numArestes=llistaArestes.size();
      double pixelsTotals = (width * height)-numArestes;
      double nPixels = pixelsTotals/longitudTotal;

      //creem el multimap
      multimap<int, int> mm;

      per i de 0 a height fer
            mm.inserir(width,i);
      fper

      //col·loquem les arestes a la parametrització
      Aresta *eActual=NULL;
      double longitudActual=0;
      int numPixels=0;
      per i de 0 a numArestes-1 fer
            //calculem els píxels que necessita l'aresta
            eActual = llistaArestes[i];
            longitudActual = eActual->getLongitud();
            numPixels = (int)ceil(longitudActual * nPixels);

            si no queden files disponibles llavors
                  mostrar missatge error i sortir;
            fsi

            //si els píxels que corresponen a l'aresta superen el width
            //la posem en una fila buida o a la que tingui més espai
            int numFila=-1, espai=0;
            si numPixels>=width llavors
                  numFila=obtenirFilaBuida();
```

[2] Multimap: Seqüència de parells (clau, valor) ordenats, permetent la duplicitat de claus.

```
                    si numFila>-1 llavors
                          //guardem els punts de parametritzacio
                          eActual->setPInici(0,numFila);
                          eActual->setPFinal(width,numFila);

                          //eliminem la fila del multimap
                          mm.eliminar(numFila);
                    altrament
                          numFila=mm.obtenirFilaAmbMesEspai();
                          espai=mm.obtenirEspai(numFila);

                          //guardem els punts de parametritzacio
                          eActual->setPInici((width - espai),numFila);
                          eActual->setPFinal(width,numFila);

                          //eliminem la fila del multimap
                          mm.eliminar(numFila);
                    fsi
              altrament
                    //busquem una fila amb prou espai
                    numFila=mm.obtenirFilaAmbProuEspai(numPixels);

                    si no queden files amb prou espai llavors
                          mostrar missatge error i sortir;
                    fsi

                    espai=mm.obtenirEspai(numFila);

                    //guardem els punts de parametritzacio
                    eActual->setPInici((width - espai),numFila);
                    eActual->setPFinal((width-espai)+numPixels-1,numFila);

                    //actualitzem la fila del multimap
                    mm.eliminar(numFila);
                    mm.inserir(espai-numPixels, numFila);
              fsi
        fper
ffunció
```

Es pot veure un exemple de visualització de la parametrització de la xarxa en la *figura 6.13*.

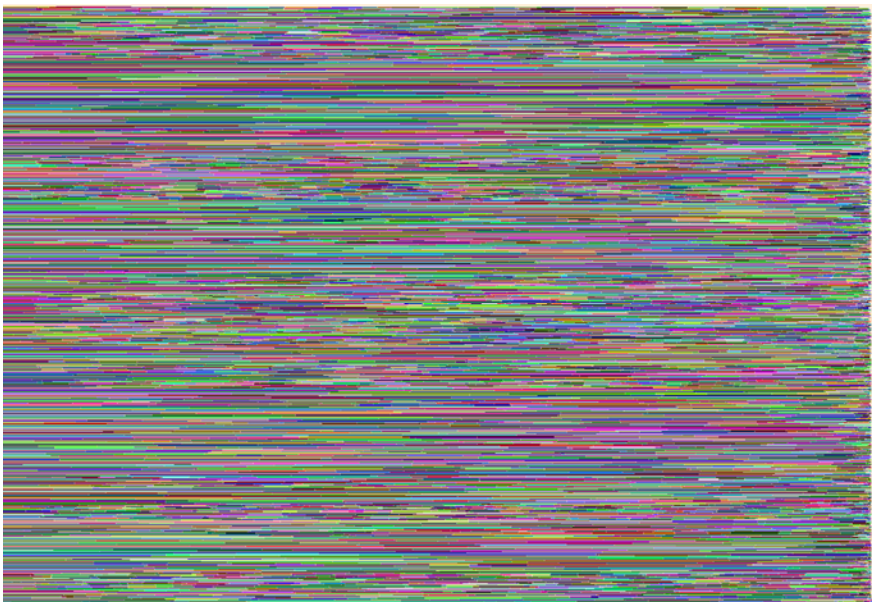


Figura 6.13: Parametrització de la xarxa

6.7. Frame Buffer Object

Amb l'objectiu de poder resoldre els diferents problemes i fer els càlculs independentment del què tenim a pantalla s'ha utilitzat el Frame Buffer Object .

El Frame Buffer Object ens permet renderitzar i utilitzar textures d'una forma molt còmode, tal i com si utilitzéssim el buffer de pantalla.

Per a poder-lo utilitzar fàcilment s'han creat 4 mètodes:

- **crearFBO**(width,height). Aquest mètode permet crear un frame buffer de la mida desitjada.
- **netejarFBO**. Aquest mètode inicialitza i neteja el que pugui haver-hi en el buffer. Es neteja la profunditat a 1 i el color a negre.
- **activarFBO**. Aquest mètode permet treballar amb el frame buffer.
- **desactivarFBO**. Aquest mètode desactiva el frame buffer.
- **eliminarFBO**. Aquest mètode eliminar el frame buffer creat.

6.8. Algorisme de Dijkstra

Per a poder calcular el camí mínim entre diferents punts de la xarxa utilitzem l'algorisme de camins mínims de Dijkstra.

Per a poder començar l'algorisme cal indicar un punt a partir del qual es calcularà la distància mínima fins a tots els vèrtexs de la xarxa. Aquest punt serà una facilitat o un punt de consulta, segons sigui necessari, que es passarà per paràmetre a la funció. El punt de consulta o la facilitat inicial es troba sobre una aresta, així que s'agafaran els dos vèrtexs d'aquesta aresta i se'ls assignarà el cost fins a la facilitat o punt de consulta inicial, i el vèrtex amb menys cost serà el que començarà a propagar. La resta de vèrtexs s'inicialitzen.

Tot seguit, es crea un multimap (i els iteradors corresponents) per ordenar segons el cost els vèrtexs als quals es pot arribar però que encara no tenen el cost mínim definitiu, així es podrà trobar fàcilment el vèrtex amb cost menor. S'afegiran els dos vèrtexs inicials al multimap.

Mentre el multimap no estigui buit:

- Agafar el vèrtex del multimap amb cost mínim.
- Per a cada aresta que surt del vèrtex:
  - Buscar el vèrtex de l'altre costat de l'aresta.
  - Si des del vèrtex actual es pot arribar a l'altre vèrtex amb menor cost del que tenia assignat, actualitzar aquest vèrtex assignant-li el cost d'arribar-hi(cost vèrtex actual + pes de l'aresta) i indicant que el seu antecessor és el vèrtex actual. Afegir aquest vèrtex al multimap si no hi era o actualitzar-lo si ja hi era.

- Eliminar el vèrtex actual del multimap.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció algorismeDijkstra(bool esFacilitat, Facilitat *facilitatActual, bool
esPuntConsulta, PuntConsulta *puntConsultaActual){

	//punt amb cost maxim
	double costMaxim=0;

	//agafem els dos vèrtexs de l'aresta
	Vertex *vInici=NULL, *vFinal=NULL;

	si esFacilitat llavors
		vInici = facilitatActual->getAresta()->getVertexOrigen();
		vFinal = facilitatActual->getAresta()->getVertexDesti();
		vInici->setCostCamiMinim(vInici.distancia(facilitatActual));
		vFinal->setCostCamiMinim(vFinal.distancia(facilitatActual));
	altrament si esPuntConsulta llavors
		vInici = puntConsultaActual->getAresta()->getVertexOrigen();
		vFinal = puntConsultaActual->getAresta()->getVertexDesti();
		vInici->setCostCamiMinim(vInici.distancia(puntConsultaActual));
		vFinal->setCostCamiMinim(vFinal.distancia(puntConsultaActual));
	fsi

	//inicialitzem els vèrtexs
	per i de 0 a llistaVertexs.size()-1 fer
		llistaVertexs[i]->setVAnt(NULL);
		llistaVertexs[i]->setCostCamiMinim(MAX_UNSIGNED);
	fper

	//creem el multimap
	multimap<double, Vertex*> mm;
	mm.inserir(vInici->getCostCamiMinim(), vInici);
	mm.inserir(vFinal->getCostCamiMinim(), vFinal);

	//mentre el multimap no estigui buit anem calculant
	Vertex *vertexActual=NULL;
	double costVActual=-1;
	mentre mm.noBuit() fer
		//agafem el vèrtex que té cost mínim
		costVActual=mm.obtenirCostMinim();
		vertexActual=mm.obtenirVertexCostMinim();

		//per a cada aresta que surt del vèrtex actual...
		vector<Aresta *> arestesActuals=vertexActual->getArestes();
		per i de 0 a arestesActuals.size()-1 fer
			//agafem l'aresta que toca i la seva longitud
			Aresta *arestaActual=arestesActuals[i];
			double pesAresta=arestaActual->getLongitud();

			//busquem el cost mínim de l'altre vèrtex de l'aresta
			Vertex *vAux=obtenirAltreVertex(arestaActual,
					vertexActual);
			double costVAux=vAux->getCostCamiMinim();

			//mirem si arribem a l'altre vèrtex amb menor cost
			si ((costVAux > costVActual + pesAresta)) llavors
				//afegim l'element al multimap o el modifiquem
				si costVertex2 == MAX_UNSIGNED)llavors
					mm.inserir((costVActual+pesAresta), vertex2);
				altrament //busquem l'element i el modifiquem
					int numFila=mm.obtenirPosicioVertex(vAux);
					mm.eliminar(numFila);
					mm.inserir((costVActual+pesAresta), vAux);
```

```
                        fsi

                        //canviem el cost i l'antecessor
                        vAux->setCostCamiMinim(costVActual+pesAresta);
                        vaux->setVAnt(vertexActual);
                    fsi
                fper

                //eliminem el vèrtex actual
                mm.eliminarPrimerElement();
            fmentre
ffunció
```

6.9. Funció camp de distàncies

La funció camp de distàncies determina quina és la distància des d'una determinat objecte fins a qualsevol punt de la xarxa.

Primer cal calcular Dijkstra i després podrem calcular la funció camp de distàncies. Aquest càlcul es fa sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs. Després de calcular Dijkstra i obtenir el cost del camí mínim per arribar a tots els vèrtexs del graf, caldrà discretitzar la funció distància i així trobar el valor als punts interiors de l'aresta per interpolació a partir dels valors que tenim als vèrtexs.

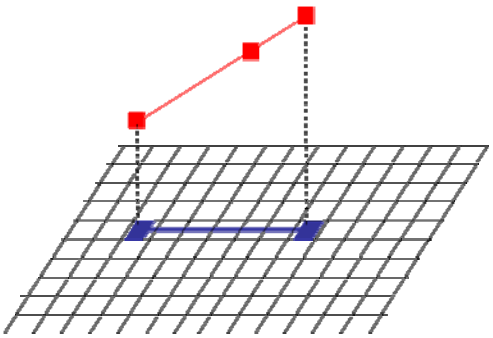


Figura 6.14: Funció distància

S'utilitzarà el buffer de profunditat per a guardar el cost del camí mínim per a cada píxel. Cal destacar que la profunditat va de 0 a 1, per tant, tenint en compte el valor màxim al qual pot arribar el cost, s'haurà de canviar de rang els valors de cost que ens resulten de calcular Dijkstra. El buffer de color també s'utilitzarà, la idea és pintar la xarxa com un degradat de color verd on el verd fosc indica més proximitat a la facilitat o punt de consulta inicial i el color verd clar indica més llunyania de la facilitat o punt de consulta inicial. Per a poder calcular la profunditat i el color s'utilitzarà el Pixel Shader.

Primer de tot, s'activa i es neteja el frame buffer. Seguidament, a través del llenguatge Cg, caldrà activar i carregar el Pixel Shader necessari. Al netejar el buffer ja es té el color a negre i la profunditat a 1, per tant, només faltarà activar el test de profunditat amb la funció menor o igual, així es consideraran tots les profunditats menors o iguals que 1. Ara ja només cal pintar cada una de les arestes del graf passant al Pixel Shader els paràmetres necessaris. Un cop pintades ja tenim la feina feta i només cal llegir els pixels de profunditat i color per a guardar-los. Finalment, caldrà desactivar el Pixel Shader i el frame buffer.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció campDistancies(bool esFacilitat, bool esPuntConsulta)

	//fbo
	crearFBO(); activarFBO(); netejarFBO();

	//activem el Pixel Shader
	carregarActivarPixelShader('campDistancies');
	glEnable(GL_DEPTH_TEST);
	glDepthFunc(GL_LEQUAL);

	//pintem les arestes
	double costMaxim=getCostMaximDijkstra();
	per i de 0 a llistaArestes.size()-1 fer
		pintarAresta(llistaArestes[i],costMaxim);
	fper

	//llegim i guardem el resultat
	vector<float> profunditat(width*height,0);
	vector<float> colorVerd(width*height,0);
	colorVerd=llegirPixels(GREEN);
	profunditat=llegirPixels(DEPTH);

	si esFacilitat llavors
		guardaCampProfunditatFacilitats(profunditat);
		guardaCampColorFacilitats(colorVerd);
	altrament si esPuntConsulta llavors
		guardaCampProfunditatPuntsConsulta(profunditat);
		guardaCampColorPuntsConsulta(colorVerd);
	fsi

	//desactivem pixel shader i fbo
	desactivarPixelShader();
	desactivarFBO(); eliminarFBO();
ffunció
```

Els camps de distàncies s'utilitzen contínuament per a calcular diferents funcionalitats i resoldre els problemes de proximitat. Per aquest motiu s'ha decidit que, un cop calculats, es guardaran en memòria i no es tornaran a calcular si no hi ha canvis. Aquest fet té dos efectes: un de positiu i un de negatiu. El negatiu és que degut a la gran quantitat d'informació guardada es perd molta memòria i això pot donar problemes si la màquina amb la qual es treballa té poca memòria o si es fan càlculs amb un nombre molt elevat d'objectes. El fet positiu és que al no haver de calcular els camps de distàncies cada vegada que s'utilitzen fa estalviar molt de temps de càlcul i, per tant, la resolució dels problemes es fa més ràpidament.

En la *figura 6.15* es pot veure un exemple del resultat que s'obté sobre la parametrització al calcular la funció camp de distàncies.

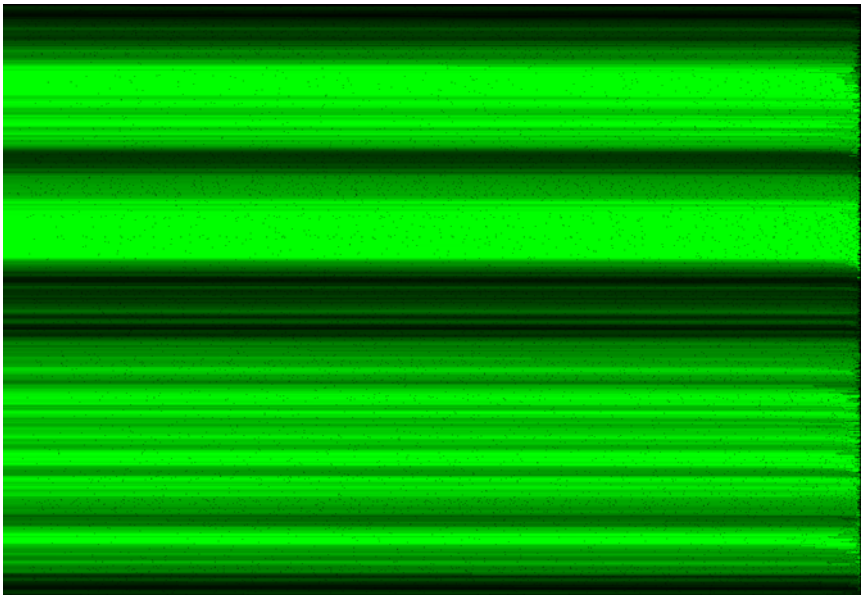


Figura 6.15: Funció camp de distàncies sobre la parametrització de la xarxa

6.10. Diagrames de Voronoi

Els diagrames de Voronoi representen informació de proximitat sobre un conjunt de punts.

6.10.1. Diagrames de Voronoi proper i llunyà

El càlcul del diagrama de Voronoi proper i del diagrama de Voronoi llunyà és molt similar. Veurem com es calculen aquests diagrames de Voronoi respecte de les facilitats de la xarxa. Es farà sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs.

S'utilitzarà el buffer de profunditat per a guardar el cost del camí mínim per a cada píxel i el buffer de color per pintar el píxel del color de la facilitat a la qual fa referència, és a dir, la facilitat que té més propera en el cas del Voronoi proper o la facilitat que té més llunyana en el cas del Voronoi llunyà. Així, abans de poder calcular el Voronoi cal que cada facilitat tingui assignat un color diferent. Per a poder calcular la profunditat i el color s'utilitzarà el Pixel Shader.

Primer de tot, cal calcular els camps de distàncies de les facilitats si encara no estan calculats i activar i netejar el frame buffer. Tot seguit, caldrà crear una textura que s'utilitzarà per a poder pintar els camps de distàncies. Seguidament caldrà activar i

carregar el Pixel Shader necessari, activar el test de profunditat i mirar si volem calcular el Voronoi proper o el llunyà. Si es vol calcular el proper s'haurà de netejar la profunditat a 1 i activar la funció menor o igual, d'aquesta manera només es guardarà per a cada píxel el valor menor de profunditat, que serà la distància a la facilitat més propera. Si pel contrari es vol calcular el llunyà s'haurà de netejar la profunditat a 0 i activar la funció major o igual, d'aquesta manera només es guardarà per a cada píxel el valor major de profunditat, que serà la distància a la facilitat més llunyana. Seguidament, s'ha de pintar el camp de distàncies de cada una de les facilitats passant al Pixel Shader la informació necessària. Per a fer-ho s'obté el camp, s'assigna a la textura i es pinta amb el color de la facilitat. Ara ja es pot llegir i guardar el resultat. Finalment, caldrà desactivar el Pixel Shader, les textures i el frame buffer.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularVoronoi(bool calcularVoronoiProper, bool calcularVoronoiLLunya)

	//si els camps distancies no estan calculats els calculem
	si no campsDistanciesCalculats llavors
		calcularCampsDistancies();
	fsi

	//fbo
	crearFBO(); activarFBO(); netejarFBO();

	//textura
	activarTextures();
	textura=crearTextura();

	//activem el Pixel Shader
	carregarActivarPixelShader('voronoi');
	glEnable(GL_DEPTH_TEST);
	si calcularVoronoiProper llavors
		glClearDepth(1);
		glDepthFunc(GL_LEQUAL);
	altrament si calcularVoronoiLlunya llavors
		glClearDepth(0);
		glDepthFunc(GL_GEQUAL);
	fsi
	glClear(GL_DEPTH_BUFFER_BIT);

	//pintem el camp de cada una de les facilitats
	per i de 0 a facilitats.size()-1 fer
		assignarTextura(textura,obtenirCampFacilitat(i));
		assignarColorFacilitat(i);
		pintar(i);
	fper

	//guardem Voronoi
	vector<float> color((width*height)*3,0);
	vector<float> profunditat(width*height,0);
	color=llegirPixels(RGB);
	profunditat=llegirPixels(DEPTH);

	si calcularVoronoiProper llavors
		guardaVoronoiProperColor(color);
		guardaVoronoiProperProfunditat(profunditat);
	altrament si calcularVoronoiLlunya llavors
		guardaVoronoiLlunyaColor(color);
		guardaVoronoiLlunyaProfunditat(profunditat);
	fsi
```

```
        //desactivem
        desactivarPixelShader();
        desactivarTextures();
        desactivarFBO(); eliminarFBO();
ffunció
```

Els diagrames de Voronoi proper i llunyà es guarden en memòria ja que s'utilitzen en altres funcionalitats. Al ser només dos, pràcticament no ocupen memòria i tenir-los guardats pot estalviar temps de càlcul.

En la *figura 6.16* es pot veure un exemple del resultat que s'obté sobre la parametrització al calcular el Voronoi proper.

Figura 6.16: Diagrama de Voronoi sobre la parametrització de la xarxa

6.10.2. Diagrama de Voronoi d'ordre k

El càlcul del diagrama de Voronoi d'ordre k es basarà en els diagrames de Voronoi calculats en l'apartat anterior, fent algunes modificacions. La idea és que a partir dels camps de distàncies, enlloc d'obtenir només la primera capa, s'anirà fent peeling per a obtenir totes les capes fins a arribar a la capa k. Es farà sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs.

S'utilitzarà el buffer de profunditat per a guardar el cost del camí mínim per a cada píxel i el buffer de color per pintar el píxel del color de la facilitat a la qual fa referència, és a dir, la k facilitat que té més propera. Així, abans de poder calcular el Voronoi k cal que cada facilitat tingui assignat un color diferent. Per a poder calcular la profunditat i el color s'utilitzarà el Pixel Shader.

Primer de tot, cal calcular els camps de distàncies de les facilitats si encara no estan calculats i activar i netejar el frame buffer. Tot seguit, caldrà crear dues textures: una que s'utilitzarà per a poder pintar els camps de distàncies i l'altra que s'utilitzarà

per a poder utilitzar la capa de Voronoi calculada en el pas anterior necessària per a poder fer peeling. Seguidament caldrà activar i carregar el Pixel Shader necessari i activar el test de profunditat amb la funció menor o igual.

Tot seguit, des de la capa 1 fins a la capa k es seguiran els següents passos:

- Netejar el buffer de color i el buffer de profunditat a 1.
- Pintar el camps de distàncies de totes les facilitats del graf passant al Pixel Shader els paràmetres necessaris. La funció del Pixel Shader és molt important, ja que ha de pintar tan sols aquells píxels que tenen profunditat superior als de la textura que conté la capa k anterior, així, s'aconsegueix fer peeling capa a capa. Cal remarcar que s'ha activat el buffer de profunditat a 1 amb la funció menor o igual per tal d'aconseguir el resultat esperat.
- Llegir i guardar la capa de color actual.
- Llegir i guardar a la textura la capa de profunditat actual.

Finalment, caldrà desactivar el Pixel Shader, les textures i el frame buffer.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularVoronoiK(int k)

	//si els camps distancies no estan calculats els calculem
	si no campsDistanciesCalculats llavors
		calcularCampsDistancies();
	fsi

	//fbo
	crearFBO(); activarFBO(); netejarFBO();

	//textura
	activarTextures();
	textura1=crearTextura(); //camp de distàncies
	textura2=crearTextura(); //capa Voronoi
	assignarTextura(textura2,0); //inicialització

	//activem el Pixel Shader
	carregarActivarPixelShader('voronoiK');
	glEnable(GL_DEPTH_TEST);
	glDepthFunc(GL_LEQUAL);

	//peeling
	per i de 1 a k fer
		glClearDepth(1);
		glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

		//pintem el camp de cada una de les facilitats
		per i de 0 a facilitats.size()-1 fer
			assignarTextura(textura1,obtenirCampFacilitat(i));
			assignarColorFacilitat(i);
			pintar(i);
		fper

		//llegim i guardem la capa actual
		vector<float> color((width*height)*3,0);
		color=llegirPixels(RGB);
		guardarVoronoiK(color);

		//copiem a la textura la capa actual
```

```
            vector<float> profunditat(width*height,0);
            profunditat=llegirPixels(DEPTH);
            assignarTextura(textura2,profunditat);
      fper

      //desactivem
      desactivarPixelShader();
      desactivarTextures();
      desactivarFBO(); eliminarFBO();
ffunció
```

Com a resultat del diagrama de Voronoi k guardem les k capes de color calculades. En el moment de visualitzar el resultat, l'usuari té dues opcions:

- **Veure el diagrama de Voronoi d'ordre k**. Aquesta opció correspon a veure les k capes obtingudes pintades utilitzant transparència.

- **Veure la capa k**. Aquesta opció correspon a veure tan sols la capa k.

6.11. Càlcul del cercle

Es farà el càlcul del cercle i la localització de les facilitats que queden al seu interior. Aquest càlcul es fa sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs.

El cercle es farà amb centre a una facilitat escollida per l'usuari i amb un radi donat. S'utilitzarà el buffer de profunditat per a guardar el cost del camí mínim per a cada píxel que es trobi dins la zona del cercle i el buffer de color per pintar aquests píxels de color vermell. Per a poder calcular la profunditat i el color s'utilitzarà el Pixel Shader.

Primerament caldrà calcular Dijkstra i el camp de distàncies de la facilitat escollida com a centre del cercle. També es pot activar i netejar el frame buffer i crear una textura que s'utilitzarà per a poder pintar el camp de distàncies de la facilitat. Tot seguit, es necessita passar el radi a l'interval 0-1 perquè sigui comparable amb els valors guardats del camp de distàncies. Seguidament caldrà activar i carregar el Pixel Shader necessari i activar el test de profunditat amb la funció menor o igual. És molt important netejar el buffer de profunditat amb el radi del cercle, això farà que tan sols es pintin aquells píxels que estan per sota d'aquest radi. Seguidament, s'assigna el color vermell que és amb el que es vol pintar els píxels que pertanyen al cercle i es pinta el camp de distàncies de la facilitat. Ja es pot guardar el cercle i calcular-ne les facilitats que hi queden dins. Finalment, caldrà desactivar el Pixel Shader, les textures i el frame buffer.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularCercle(int numFacilitat, double distancia)

	//camp de distàncies de la facilitat
	algorismeDijkstra(cert, facilitats[numFacilitat], fals, NULL);
	calcularCampDistancies(cert, fals);

	//fbo
	crearFBO(); activarFBO(); netejarFBO();

	//textura
	activarTextures();
	textura=crearTextura();
	assignarTextura(textura,obtenirCampFacilitat(numFacilitat));

	//radi del cercle
	double distanciaFinal=distancia/obtenirCostMaximDijkstra();

	//activem el Pixel Shader
	carregarActivarPixelShader('cercle');
	glEnable(GL_DEPTH_TEST);
	glDepthFunc(GL_LEQUAL);
	glClearDepth(distanciaFinal);
	glClear(GL_DEPTH_BUFFER_BIT);

	//pintem
	assignarColor(vermell);
	pintar();

	//guardem cercle
	vector<float> color((width*height)*3,0);
	color=llegirPixels(RGB);
	guardaCercle(color);

	//busquem les facilitats resultants
	vector<Facilitat*> facilitatsCercle=obtenirFacilitatsCercle();

	//desactivem
	desactivarPixelShader();
	desactivarTextures();
	desactivarFBO(); eliminarFBO();
ffunció
```

6.12. Punts d'interès

Els punts d'interès representen punts estratègics sobre la xarxa. Es calculen 4 punts: 1-Center, 1-Median, 1-Center Obnoxious i 1-Median Obnoxious. Aquest càlcul es fa sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs.

6.12.1. 1-Center

El punt d'interès 1-Center ens retorna el punt de la xarxa que minimitza la màxima distància a les facilitats. És un punt que s'utilitza per a col·locar un nou servei que la gent vol tenir a prop.

El punt 1-Center el podem calcular fàcilment a partir del diagrama de Voronoi llunyà. Així, el primer que fem és calcular aquest diagrama. Seguidament, el que cal fer és buscar el píxel que té el valor de profunditat menor. Un cop obtingut aquest píxel de la parametrització cal fer la conversió i trobar a quina posició de la xarxa correspon. Aquesta posició ja és el punt 1-Center.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcular1Center()
       //calculem Voronoi llunyà
       calcularVoronoi(fals, cert);

       //busquem la posició
       float valor1Center=voronoiLlunyaProfunditat[0];
       int posicio1Center=0;

       per i de 1 a voronoiLlunyaProfunditat.size()-1 fer
              float valorActual=voronoiLlunyaProfunditat[i];
              si (valorActual<valor1Center i valorActual!=0.0) o
                 (valor1Center==0.0 i valorActual!=0.0) llavors
                     valor1Center=valorActual;
                     posicio1Center=i;
              fsi
       fper

       //trobem l'aresta on està aquest punt
       Aresta *e1Center=trobarAresta(posicio1Center);

       //trobem el punt exacte dins l'aresta
       Facilitat *f1Center=trobarPunt(e1Center, posicio1Center);
       f1Center->setNom("[1-Center]");
ffunció
```

6.12.2. 1-Center Obnoxious

El punt 1-Center Obnoxious ens retorna el punt de la xarxa que maximitza la distància mínima a les facilitats. És un punt que s'utilitza per a col·locar un nou servei que la gent vol tenir lluny.

El punt 1-Center Obnoxious el podem calcular fàcilment a partir del diagrama de Voronoi proper. Així, el primer que fem és calcular aquest diagrama. Seguidament, el que cal fer és buscar el píxel que té el valor de profunditat major. Un cop obtingut aquest píxel de la parametrització cal fer la conversió i trobar a quina posició de la xarxa correspon. Aquesta posició ja és el punt 1-Center Obnoxious.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcular1CenterObnoxious()
       //calculem Voronoi proper
       calcularVoronoi(cert, fals);

       //busquem la posició
       float valor1CenterOb=voronoiProperProfunditat[0];
       int posicio1CenterOb=0;

       per i de 1 a voronoiProperProfunditat.size()-1 fer
             float valorActual=voronoiProperProfunditat[i];
             si (valorActual>valor1CenterOb i valorActual!=1.0) o
                (valor1CenterOb==1.0 i valorActual!=1.0) fer
                    valor1CenterOb=valorActual;
                    posicio1CenterObs=i;
             fsi
       fper

       //trobem l'aresta on està aquest punt
       Aresta *e1CenterOb=trobarAresta(posicio1CenterOb);

       //trobem el punt exacte dins l'aresta
       Facilitat *f1CenterOb=trobarPunt(e1CenterOb, posicio1CenterOb);
       f1CenterO->setNom("[1-Center Obnoxious]");
ffunció
```

6.12.3. 1-Median i 1-Median Obnoxious

El punt 1-Median ens retorna el punt de la xarxa que minimitza la suma de distàncies a les facilitats. És un punt que s'utilitza per a col·locar un nou servei que la gent vol tenir a prop. Pel contrari el punt 1-Median Obnoxious ens retorna el punt de la xarxa que maximitza la suma de distàncies a les facilitats. És un punt que s'utilitza per a col·locar un nou servei que la gent vol tenir lluny.

S'utilitzarà el buffer de profunditat per a guardar la suma de profunditats de cada un del camps de distàncies de les facilitats de la xarxa. A partir d'aquesta suma ja podrem trobar els punt 1-Median i 1-Median Obnoxious. Per a poder calcular la suma de profunditats s'utilitzarà el Pixel Shader.

Primer s'activa i es neteja el frame buffer i es creen dues textures: la primera s'utilitzarà per a poder pintar cada una dels camps de distàncies i la segona s'utilitzarà per a guardar la suma d'aquests camps. Seguidament s'haurà d'activar i carregar el Pixel Shader necessari, activar el test de profunditat amb la funció major o igual per a permetre sumar tots els valors i netejar la profunditat a 0 per a poder començar la suma. Ara, per a cada facilitat es fa el següent:

- Carregar a la primera textura el camp de distàncies de la facilitat.
- Pintar passant al Pixel Shader la informació necessària.
- Llegir la suma acumulada i guardar-la a la segona textura.

Un cop finalitzat aquest procés caldrà desactivar el Pixel Shader, les textures i el frame buffer.

Ara es disposa de la suma de tots els camps de distàncies de les facilitats. Per obtenir el punt 1-Median cal buscar el píxel on el valor de la suma és menor i per obtenir el punt 1-Median Obnoxious cal buscar el píxel on el valor de la suma és major. Un cop obtinguts aquests píxel de la parametrització cal fer la conversió i trobar a quina posició de la xarxa corresponen. Aquestes posicions ja són els punts 1-Median i 1-Median Obnoxious.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcular1Median()

	//fbo
	crearFBO(); activarFBO(); netejarFBO();

	//textura
	activarTextures();
	textura1=crearTextura(); //camp de distàncies
	textura2=crearTextura(); //suma de camps
	assignarTextura(textura2,0);

	//activem el Pixel Shader
	carregarActivarPixelShader('1median');
	glEnable(GL_DEPTH_TEST);
	glDepthFunc(GL_GEQUAL);
	glClearDepth(0.0);
	glClear(GL_DEPTH_BUFFER_BIT);

	//pintem els camp distancies per tal de sumar la profunditat
	per i de 0 a facilitats.size()-1 fer
		//obtenim el camp de distàncies
		assignarTextura(textura1,obtenirCampFacilitat(i));

		//pintem
		pintar();

		//llegim i guardem la profunditat a la textura
		vector<float> profunditat(width*height,0);
		profunditat=llegirPixels(DEPTH);
		assignarTextura(textura2,profunditat);
	fper

	//desactivem
	desactivarPixelShader();
	desactivarTextures();
	desactivarFBO(); eliminarFBO();

	//busquem el valor mínim i màxim de profunditat
	float v1Median=profunditat[0];
	float v1MedianOb=profunditat[0];
	int posicio1Median=0;
	int posicio1MedianOb=0;

	per i de 1 a profunditat.size()-1 fer
		float valorActual=profunditat[i];
		si valorActual!=1.0 llavors
			si (valorActual>v1MedianOb) o (v1MedianOb==1.0) llavors
				v1MedianOb=valorActual;
				posicio1MedianOb=i;
			fsi
		fsi

		si valorActual!=0.0 llavors
			si (valorActual<v1Median) o (v1Median==0.0) llavors
```

```
                        v1Median=valorActual;
                        posicio1Median=i;
                    fsi
                fsi
        fper

        //trobem l'aresta on està aquest punt
        Aresta *e1Median=trobarAresta(posicio1Median);
        Aresta *e1MedianOb=trobarAresta(posicio1MedianOb);

        //trobem el punt exacte dins l'aresta
        Facilitat *f1Median=trobarPunt(e1Median, posicio1Median);
        f1Median->setNom("[1-Median]");
        Facilitat *f1MedianOb=trobarPunt(e1MedianOb, posicio1MedianOb);
        f1MedianOb->setNom("[1-Median Obnoxious]");
ffunció
```

6.13. Problema de proximitat k-NN

El problema de proximitat k-NN es resol sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs. L'usuari ha de seleccionar un punt de consulta i escollir si vol utilitzar totes les facilitats o només les d'un determinat tipus (gasolineres, hotels, restaurants,...). A la funció calcularKNN se li pot passar per paràmetre un punt de consulta o una facilitat inicial, això és degut a que algunes funcions necessiten fer el mateix càlcul que es fa amb el punt de consulta per a una facilitat. Ara, es suposarà que el càlcul es fa amb un punt de consulta inicial i que s'utilitzen totes les facilitats, per tal de no complicar el codi.

La base per a poder resoldre els k veïns propers és trobar el diagrama de Voronoi d'ordre k on la k ens indica el nombre de facilitats que es busquen com a resultat. Es necessiten cada una de les k capes que es calculen ja que cada una aportarà una de les facilitats resultants.

Es comença calculant el diagrama de Voronoi d'ordre k. Tot seguit, cal obtenir la posició que té el punt de consulta a la parametrització. Ara, per a cada una de les k capes de Voronoi calculades:

- Obtenir el color que té la posició del punt de consulta en la capa actual del diagrama de Voronoi.
- Trobar la facilitat que té aquest color i guardar-la al vector resultant.

S'obté com a resultat un vector amb les k facilitats més properes al punt de consulta marcat.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularKNN(int numFacilitats, PuntConsulta *qActual)( int
numFacilitats, int tipusFacilitat, bool esPuntConsulta, PuntConsulta *qActual,
bool esFacilitat, Facilitat *fActual)

        //fem el Voronoi k de les facilitats
        calcularVoronoiK(numFacilitats);
```

```
        //posició parametrització punt de consulta
        PuntParametritzacio *posicio=qActual->getPosicioParametritzacio();
        int x=posicio->getX();
        int y=posicio->getY();
        int posicioVector=((widthFinestra*y)+x)*3;

        //busquem per a cada capa quina és la facilitat més propera
        Facilitat *fAux=NULL;
        double r,g,b;
        per i de 0 a vectorVoronoi.size() fer
                //agafem el color de la facilitat
                r=vectorVoronoi[i][posicioVector];
                g=vectorVoronoi[i][posicioVector+1];
                b=vectorVoronoi[i][posicioVector+2];

                //trobem la facilitat que té aquest color i la guardem
                fAux=buscarFacilitat(r,g,b);
                facilitatskNN.afegir(fAux);
        fper
ffunció
```

6.14. Problema de proximitat Ak-NN

El problema de proximitat Ak-NN es resol sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs. L'usuari ha d'indicar el nombre k de facilitats que vol trobar, si vol utilitzar totes les facilitats per a fer els càlculs o només les d'un tipus concret (restaurants, hotels, ...) i escollir una funció per a fer el càlcul: MinMax, MinSum, MaxMin, MaxSum. Ara, es suposarà que el càlcul es fa utilitzant totes les facilitats, per tal de no complicar el codi.

La base per a poder resoldre els k veïns propers agregats són els camps de distàncies de cada una de les facilitats de la xarxa, que ens permetran trobar els k objectes millor situats respecte a la funció escollida. S'utilitzarà el buffer de profunditat, concretament s'assignarà un píxel per a cada facilitat, que guardarà informació diferent segons la funció que s'utilitzi: MinMax (guardarà el cost del camí mínim al punt de consulta més llunyà), MaxMin (guardarà el cost del camí mínim al punt de consulta més proper), MinSum o MaxSum (guardaran la suma de costos als punts de consulta). Per a poder calcular aquestes profunditats s'utilitzarà el Pixel Shader.

Primer de tot, cal calcular els camps de distàncies de les facilitats si encara no estan calculats i activar i netejar el frame buffer. Si s'utilitza la funció MinSum o MaxSum caldrà crear una textura que s'utilitzarà per anar-hi guardant la suma acumulada de les profunditats de cada facilitat als punts de consulta. Seguidament caldrà activar i carregar el Pixel Shader necessari i activar el test de profunditat. Si estem utilitzant la funció MaxMin s'ha de netejar el buffer de profunditat a 1 amb la funció menor o igual i si estem utilitzant qualsevol de les altres tres funcions s'ha de netejar el buffer de profunditat a 0 amb la funció major o igual per tal d'aconseguir el resultat desitjat.

Ara cal obtenir per a cada facilitat la distància màxima, mínima o la suma de distàncies als punts de consulta. Per a cada punt de consulta:

- Obtenim la posició del punt de consulta en la parametrització.
- Per a cada facilitat:
  - Amb el camp de distàncies de la facilitat obtenim la distància de la facilitat fins al punt de consulta actual.
  - Pintem aquesta distància passant al Pixel Shader els paràmetres necessaris. La pintarem al píxel que correspon a la facilitat.
- Si estem utilitzant la funció Minsum o MaxSum guardem la suma acumulada fins el moment a la textura.

S'obté com a resultat la distància màxima, mínima o la suma de distàncies de cada facilitat als punts de consulta. Ara cal trobar les k facilitats mínimes o màximes segons la funció escollida. Per tal de trobar-les es crearà un multimap (i els iteradors corresponents) per ordenar les distàncies resultants i saber a quina facilitat corresponen. Si la funció escollida és MinMax o MinSum s'obtindran les k primeres facilitats del multimap que són les que fan mínima la màxima distància o la suma de distàncies respectivament. Si la funció escollida és MaxMin o MaxSum s'obtindran les k últimes facilitats del multimap que són les que fan màxima la mínima distància o la suma de distàncies respectivament.

Finalment, caldrà desactivar el Pixel Shader, les textures i el frame buffer.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularAKNN(int numFacilitats, int tipusAgregats)

      //si els camps distancies no estan calculats els calculem
      si no campsDistanciesCalculats llavors
            calcularCampsDistancies();
      fsi

      //fbo
      crearFBO(); activarFBO(); netejarFBO();

      //si fem MinSum o MaxSum utilitzem una textura per sumar la profunditat
      si tipusAgregats==MinSum o tipusAgregats==MaxSum llavors
            //textura
            activarTextures();
            textura=crearTextura();
            assignarTextura(textura,0);
      fsi

      //activem el Pixel Shader
      carregarActivarPixelShader('aknn');
      glEnable(GL_DEPTH_TEST);

      si tipusAgregats==MinMax o tipusAgregats==MinSum o
         tipusAgregats==MaxSum llavors
            glDepthFunc(GL_GEQUAL);
            glClearDepth(0.0);
      altrament si tipusAgregats==MaxMin llavors
            glDepthFunc(GL_LEQUAL);
            glClearDepth(1.0);
      fsi
      glClear(GL_DEPTH_BUFFER_BIT);
      glEnable(GL_DEPTH_TEST);

      //per a cada facilitat pintem la distancia a tots els punts de consulta
```

```
PuntConsulta *qActual=NULL;
double distancia;
int x, y, posicioVector, pintarX, pintarY;
PuntParametritzacio *posicio=NULL;
per i de 0 a puntsConsulta.size()-1 fer
	qActual=puntsConsulta[i];
	posicio=qActual->getPosicioParametritzacio();
	x=posicio->getX();
	y=posicio->getY();
	posicioVector=(widthFinestra*y)+x;

	pintarX=0; pintarY=0;
	per i de 0 a facilitats.size()-1 fer
		//busquem la distancia de la facilitat al qActual
		distancia=obtenirCampProfunditat(j)[posicioVector];

		//obtenim el píxel on pintar la distància
		pintarX=obtenirXPixelFacilitat(facilitats[i]);
		pintarY=obtenirYPixelFacilitat(facilitats[i]);

		//pintem
		pintar(distancia, pintarX, pintarY);
	fper

	si tipusAgregats==MinSum o tipusAgregats==MaxSum llavors
		//copiem a la textura la capa actual
		vector<float> profunditatAcumulada(width*height,0);
		profunditatAcumulada=llegirPixels(DEPTH);
		assignarTextura(textura,profunditatAcumulada);
	fsi
fper

//guardem el resultat
vector<float> profunditat(width*height,0);
profunditat=llegirPixels(DEPTH);

//creem multimap(profunditat,posicio) per ordenar les distàncies
multimap<double, int> mm;
per i de 0 a facilitats.size()-1 fer
	mm.inserir(profunditat[i],i);
fper

// MinMax- k facilitats que fan mínima la màxima distància
// MinSum - k facilitats que fan mínima la suma de les distàncies
int j=0, numFila;
si tipusAgregats==MinMax o tipusAgregats==MinSum llavors
	mentre j<numFacilitats i j<facilitats.size() fer
		//trobem la facilitat
		int posicioVector=mm.obtenirPosicioFacilitat(1);
		facilitatsAKNN.inserir(facilitats[posicioVector]);

		//eliminem del multimap
		mm.erase(1);

		j++;
	fmentre
fsi

// MaxMin- k facilitats que fan màxima la mínima distància
// MaxSum - k facilitats que fan màxima la suma de les distàncies
si tipusAgregats==MaxMin o tipusAgregats==MaxSum llavors
	mentre j<numFacilitats i j<facilitats.size() fer
		//trobem la facilitat
		int posicioVector=mm.obtenirPosicioFacilitat(mm.size());
		facilitatsAKNN.inserir(facilitats[posicioVector]);

		//eliminem del multimap
```

```
                    mm.erase(mm.size());

                    j++;
                fmentre
        fsi

        //desactivem
        desactivarPixelShader();
        desactivarTextures();
        desactivarFBO(); eliminarFBO();
ffunció
```

6.15. Problema de proximitat Rk-NN

El problema de proximitat Rk-NN es resol sobre la parametrització de la xarxa utilitzant la potència de la GPU per accelerar els càlculs. L'usuari ha d'indicar la facilitat inicial, el valor k i si vol utilitzar totes les facilitats per a fer els càlculs o només les d'un tipus concret (restaurants, hotels, ...). Ara, es suposarà que el càlcul es fa utilitzant totes les facilitats, per tal de no complicar el codi. Cal remarcar que la resolució d'aquest problema es pot fer, com a màxim, amb 24 facilitats encara que és fàcilment ampliable a totes les facilitats de la xarxa.

La base per a poder resoldre els k veïns propers inversos és la resolució del problema k-NN i el càlcul del cercle. S'utilitzarà el buffer de color per a fer els càlculs i trobar les k facilitats resultants. Concretament, s'utilitzaran els bits de cada un dels 3 components de color R, G i B, en total 24 bits, un per a cada facilitat. Es tracta de que quan es pinta el cercle d'una facilitat s'activa el bit que li correspon i així, si pintem els cercles sumant el seu color obtindrem totes les interseccions i resoldrem el problema.

Per a cada facilitat de la xarxa:

- Es fa el càlcul del problema k-NN per a obtenir la k facilitat més propera.
- S'obté la distància des de la facilitat actual fins a aquesta facilitat.
- Es calcula el cercle utilitzant la facilitat actual com a centre i la distància obtinguda com a radi. Cal remarcar que el càlcul del cercle varia una mica del que s'ha explicat anteriorment ja que el color que s'assigna a la zona del cercle no és el vermell sinó que és el color resultant d'activar el bit assignat a la facilitat actual.

Un cop calculats els cercles per a cada una de les facilitats s'activa i es neteja el frame buffer, es crea una textura per a poder pintar cada un dels cercles i el més important, s'activa la suma de colors. Ara, cada un dels cercles calculats s'assigna a la textura i es pinta. Amb el resultat obtingut ja es poden trobar les k facilitats resultants. S'ha de buscar el color que té la facilitat que s'ha marcat inicialment. Mirant els bits de color que seran de la forma: 00001000-00100000-00000111 es pot veure que les facilitats resultants són aquelles que tenen el bit activat, en aquest exemple són les facilitats número 1, 2, 3, 14 i 20. Finalment, caldrà desactivar la suma de colors, les textures i el frame buffer.

Tot seguit es mostra el pseudocodi a alt nivell d'aquesta funcionalitat:

```
funció calcularRKNN(int numFacilitats, int tipusFacilitat)

      //per a cada facilitat busquem knn i fem el seu cercle
      per i de 0 a facilitats.size()-1 fer
            calcularKNN(numFacilitats+1, tipusFacilitat,
                        false, NULL, true, facilitats[i]);

            //obtenim la facilitat del kNN que ens interessa
            Facilitat *fAux = kFacilitatsNN[numFacilitats];

            //busquem la distancia de fActual fins a fAux
            PuntParametritzacio *pp=fAux->getPosicioParametritzacio();
            float distancia;
            distancia=obtenirCampProfunditat(i)[(pp->Y()*width)+pp->X()];

            //pintem el cercle i ja es guarda al vector rknnCercles
            cercleRKNN(i, distancia);
      fper

      //fbo
      crearFBO(); activarFBO(); netejarFBO();

      //textura
      activarTextures();
      textura=crearTextura();

      //sumem els colors
      glEnable(GL_DEPTH_TEST);
      glDepthFunc(GL_LEQUAL);
      glEnable(GL_COLOR_LOGIC_OP);
      glLogicOp(GL_OR);

      //un cop tenim tots els cercles calculats els pintem
      per i de 0 a knnCercles.size()-1 fer
            //assignem el cercle a la textura
            assignarTextura(textura,obtenirCercleRKNN(i));

            //pintem
            pintar();
      fper

      //guardem els rknn
      vector<float> color((width*height)*3,0);
      color=llegirPixels(RGB);
      guardaRKNN(color);

      //obtenim les facilitats resultat
      facilitatsRKNN=obtenirFacilitatsRKNN();

      //desactivem
      glDisable(GL_COLOR_LOGIC_OP);
      desactivarTextures();
      desactivarFBO(); eliminarFBO();
ffunció
```

7. Resultats

7.1. Visualització de resultats

A continuació es mostrarà la visualització de diferents xarxes de carreteres i dels resultats d'alguns dels càlculs que es fan.

7.1.1. Xarxes de carreteres

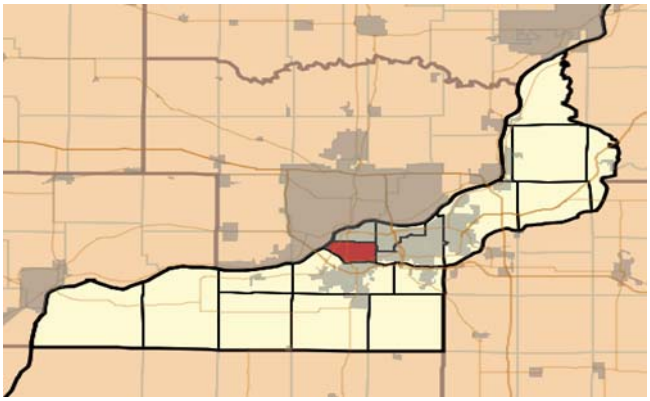


Figura 7.1: Mapa original de Rock Island

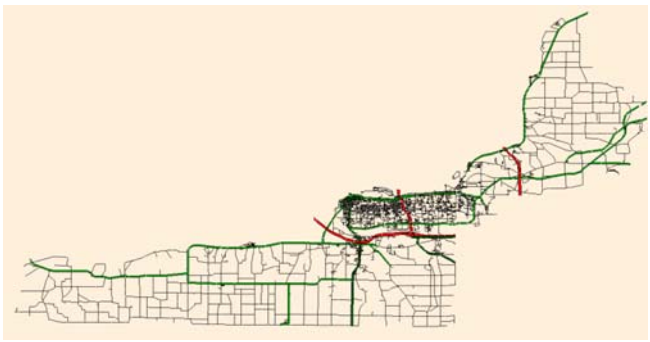


Figura 7.2: Xarxa de carreteres de Rock Island

7.1.2. Estructures de dades

7.1.2.1. Quadtree comprimit d'objectes

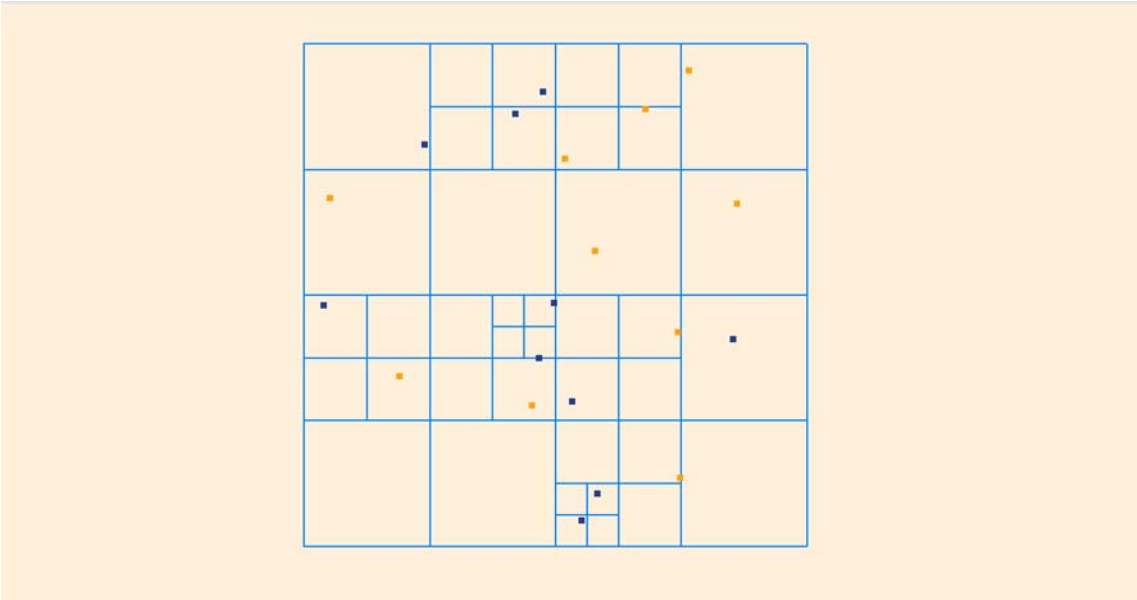


Figura 7.3: QuadTree comprimit d'objectes

7.1.2.2. Rtree d'arestes

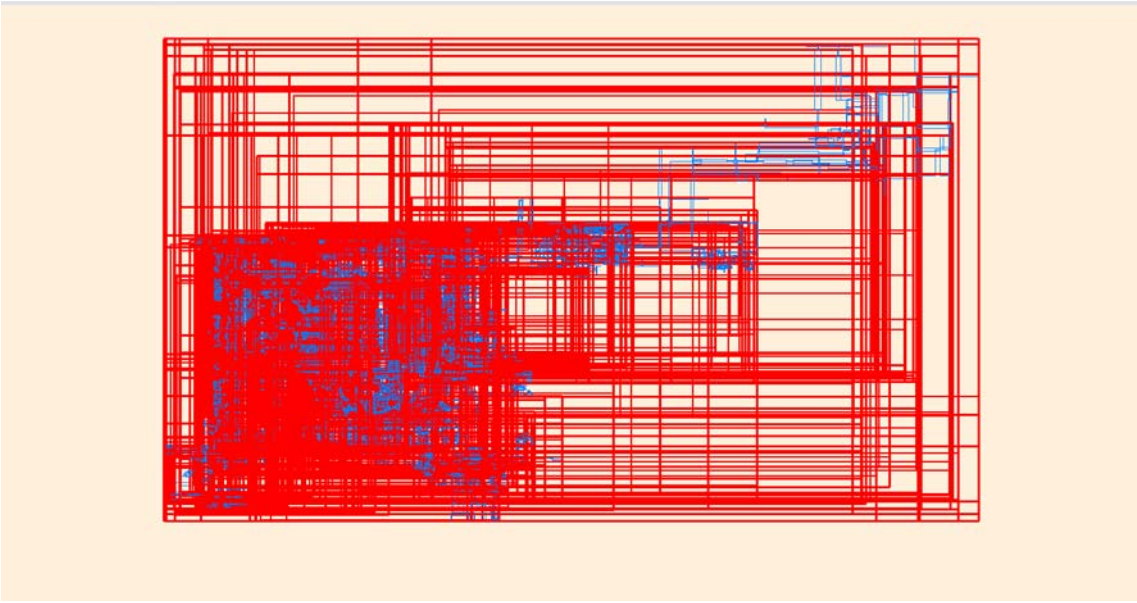


Figura 7.4: RTree d'arestes

7.1.3. Objectes: facilitats, punts de consulta i vehicles

7.1.4. Seleccionar Zona

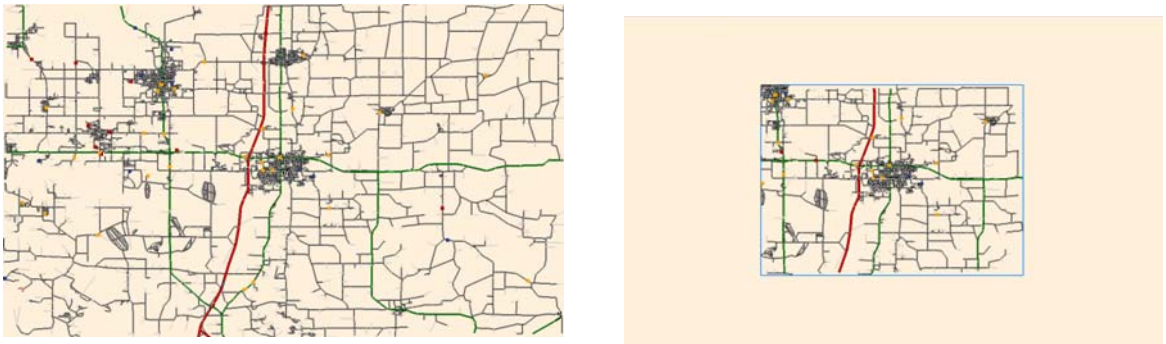


Figura 7.5: Seleccionar zona

7.1.5. Càlcul del camí mínim

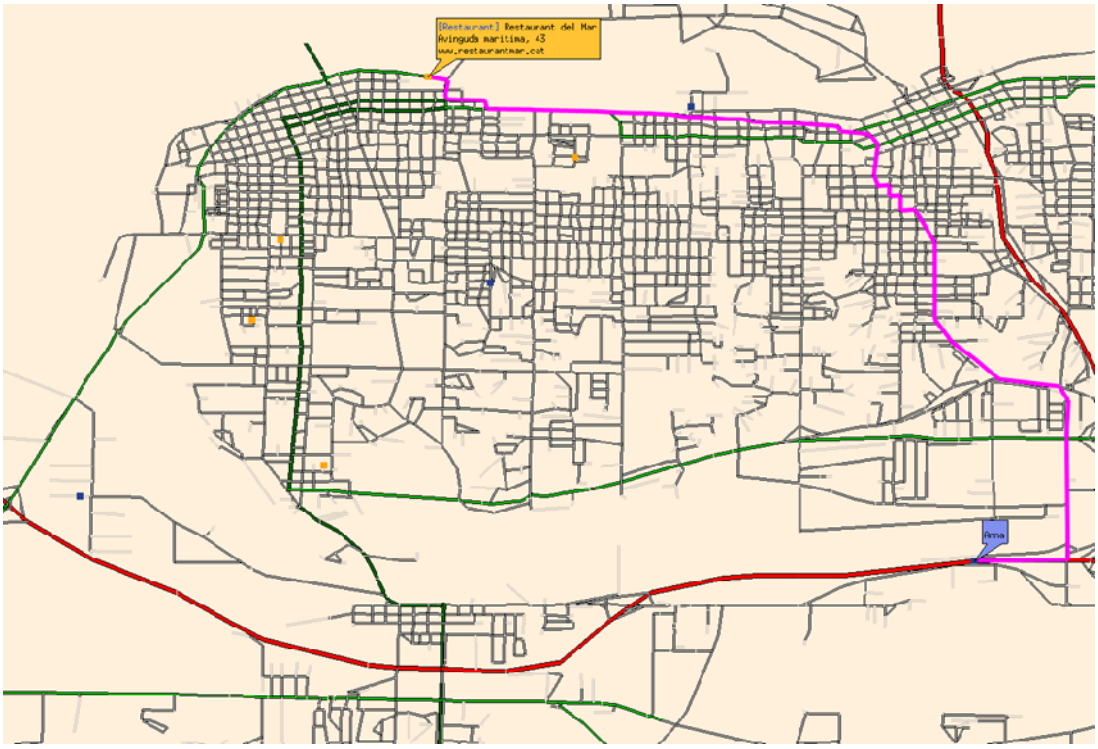


Figura 7.6: Camí mínim

7.1.6. Funció camp de distàncies

Figura 7.7: Camp de distàncies

7.1.7. Càlcul del Cercle

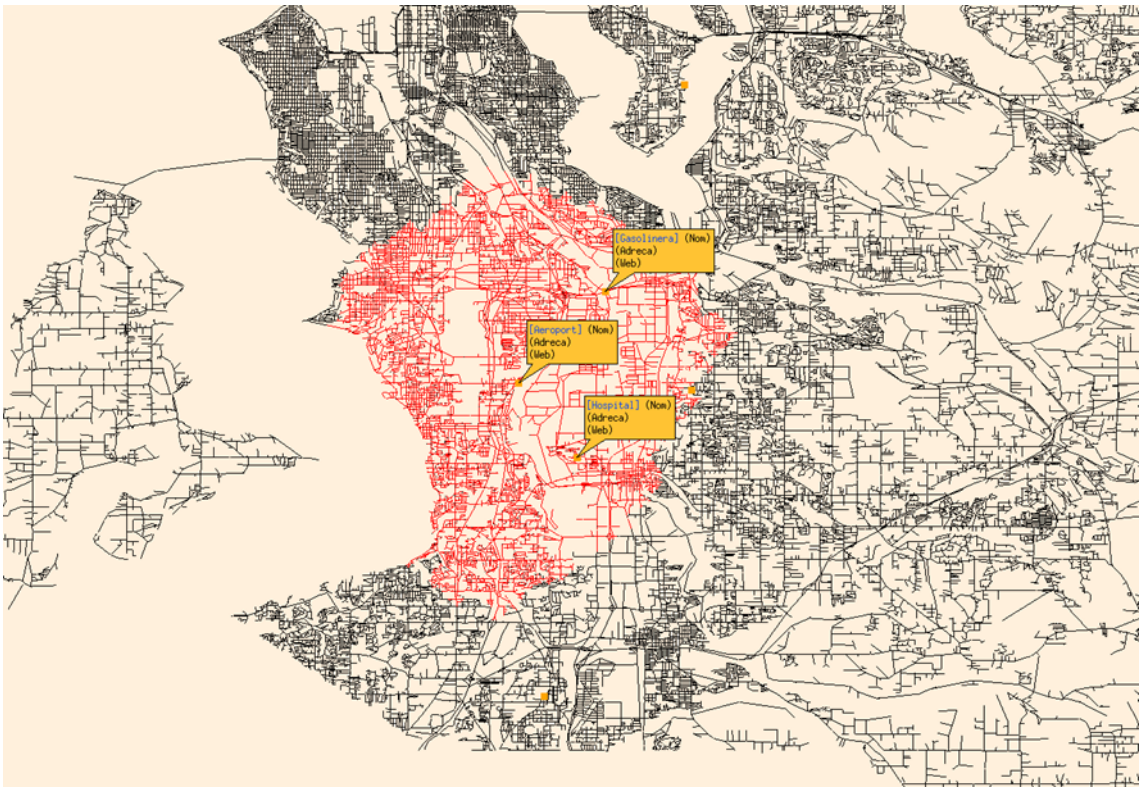


Figura 7.8: Cercle

7.1.8. Diagrames de Voronoi

7.1.8.1. Diagrama de Voronoi proper

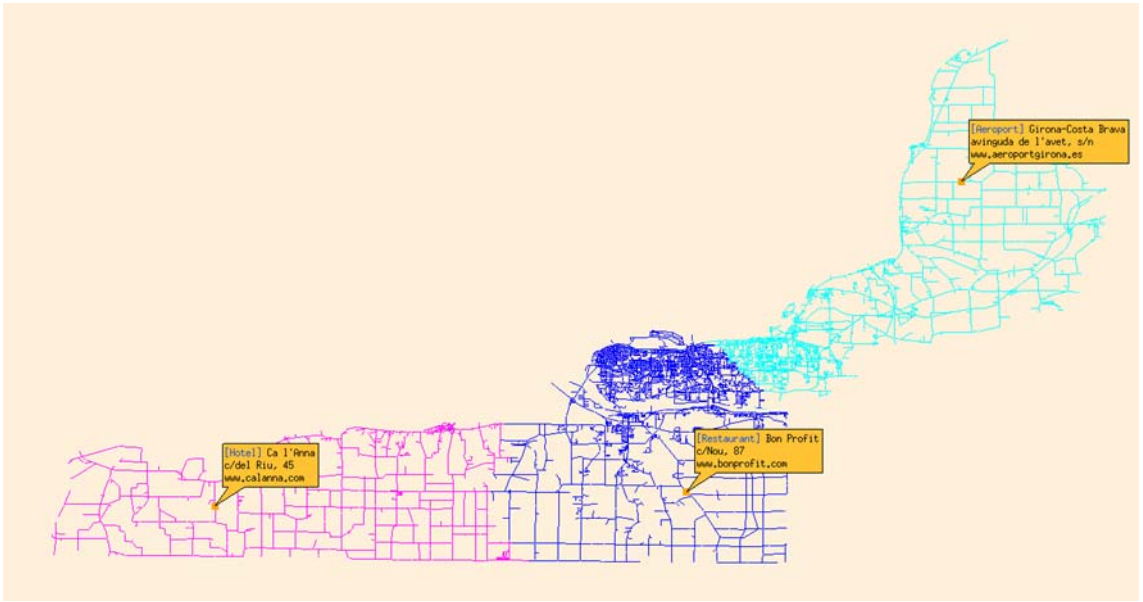


Figura 7.9: Diagrama de Voronoi proper

7.1.8.2. Diagrama de Voronoi llunyà

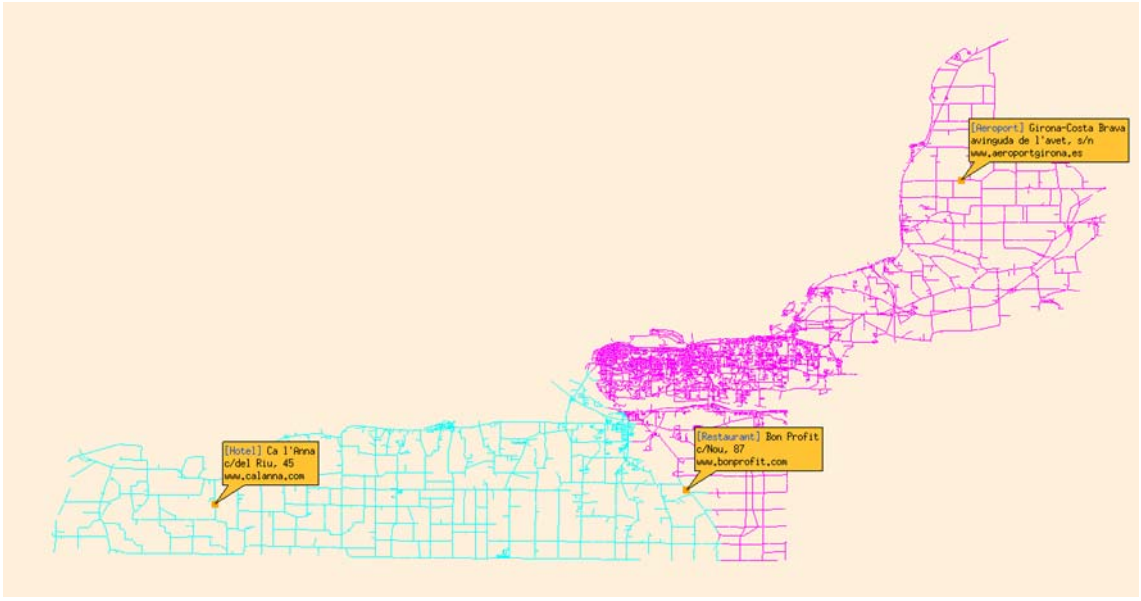


Figura 7.10: Diagrama de Voronoi llunyà

7.1.8.3. Diagrama de Voronoi ordre k

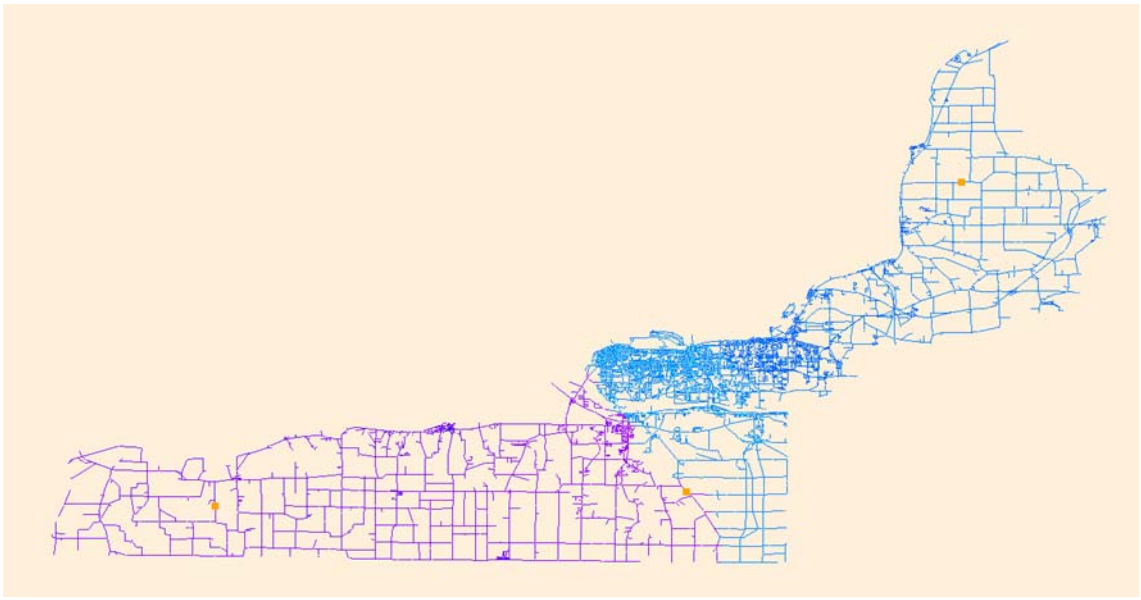


Figura 7.11: Diagrama de Voronoi d'ordre 2

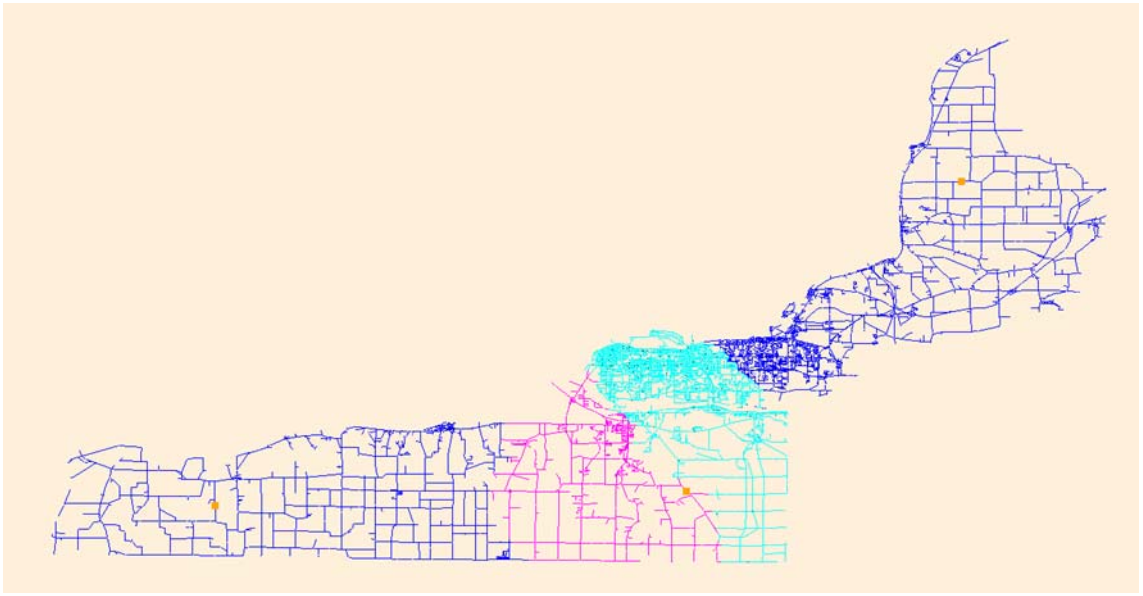


Figura 7.12: Diagrama de Voronoi capa 2

7.1.9. Punts d'interès

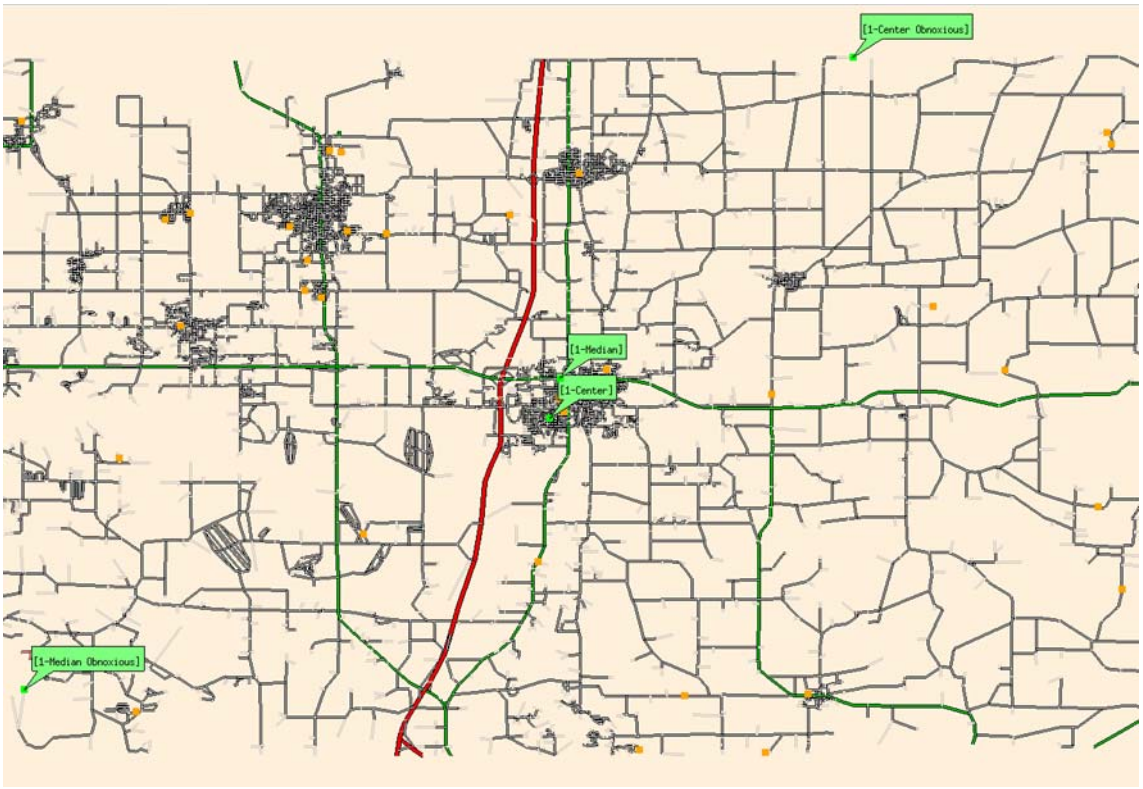


Figura 7.13: Punts d'interès

7.1.10. Problema k-NN

Tenim una xarxa amb una sèrie de facilitats de diferents tipus i un punt de consulta. Es vol obtenir la gasolinera més propera al punt de consulta.

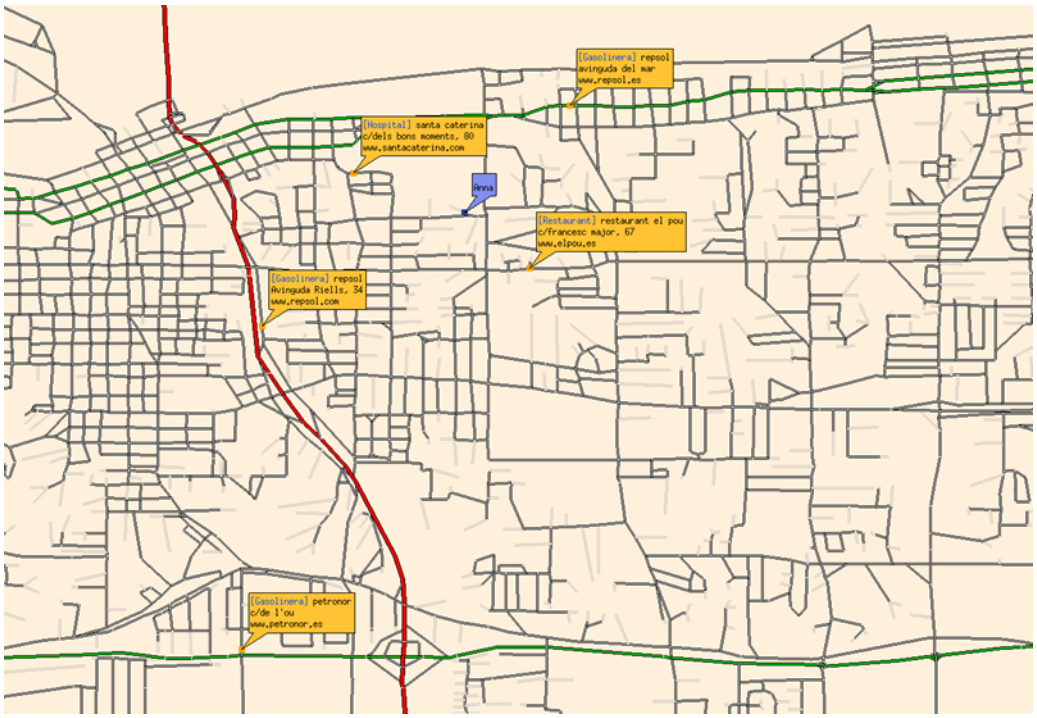


Figura 7.14: Problema k-NN: Abans del càlcul

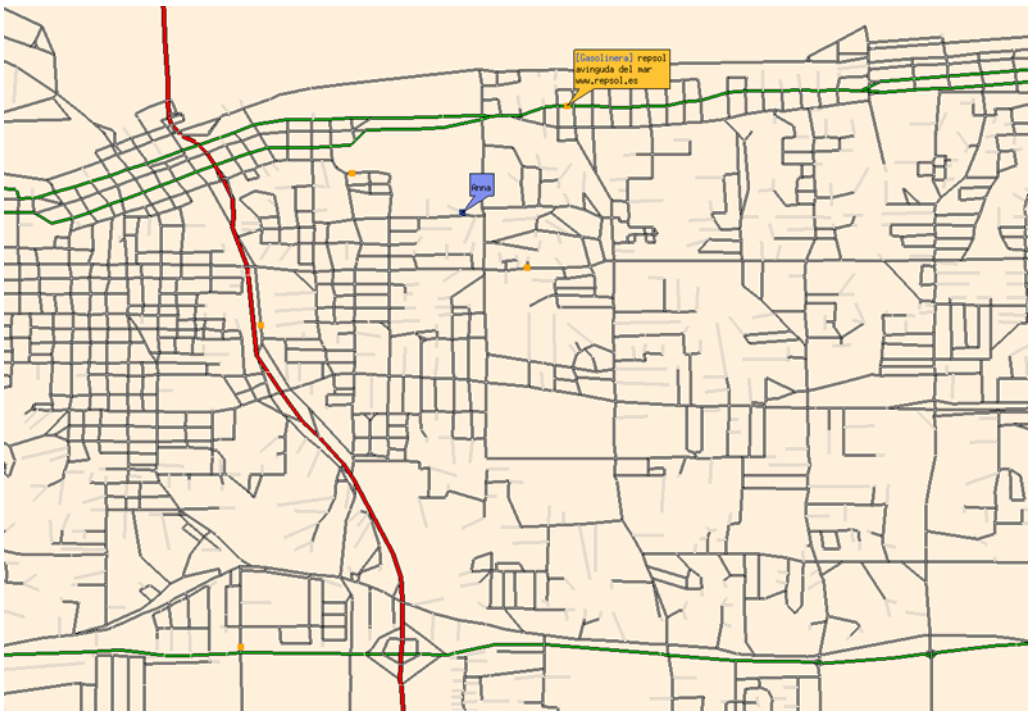


Figura 7.15: Problema k-NN: Resultat

7.1.11. Problema Rk-NN

Tenim una xarxa amb una sèrie de facilitats de diferents tipus. Es vol resoldre el problema a partir de l'aeroport i per a k=3.

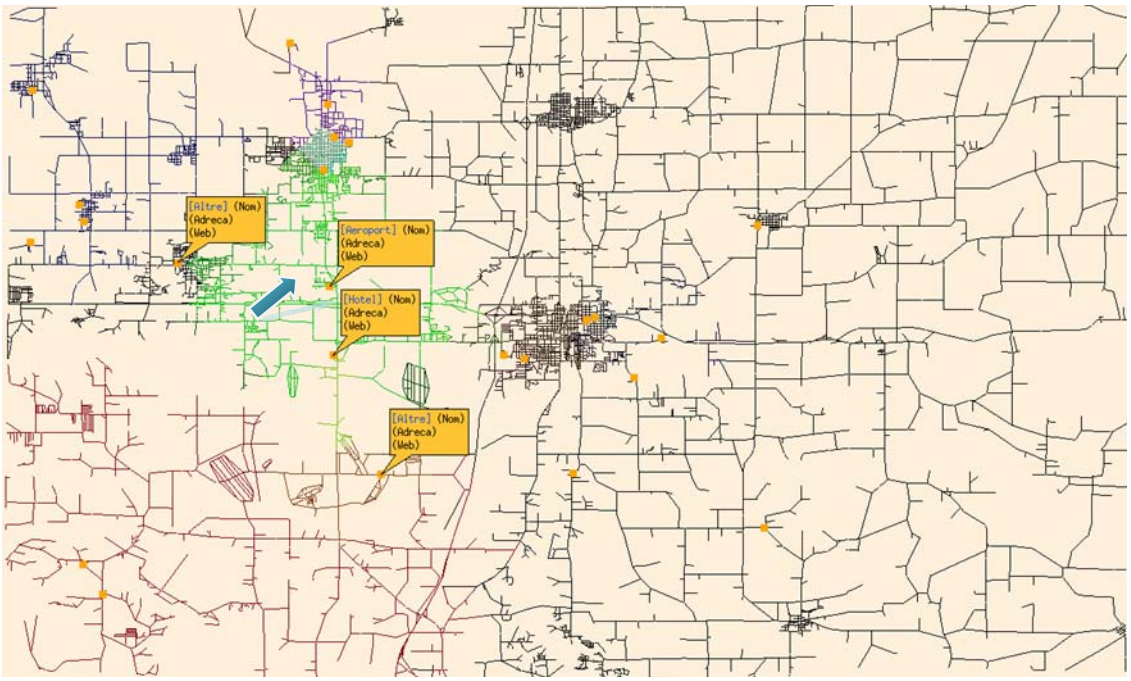


Figura 7.16: Problema Rk-NN

7.1.12. Problema Ak-NN

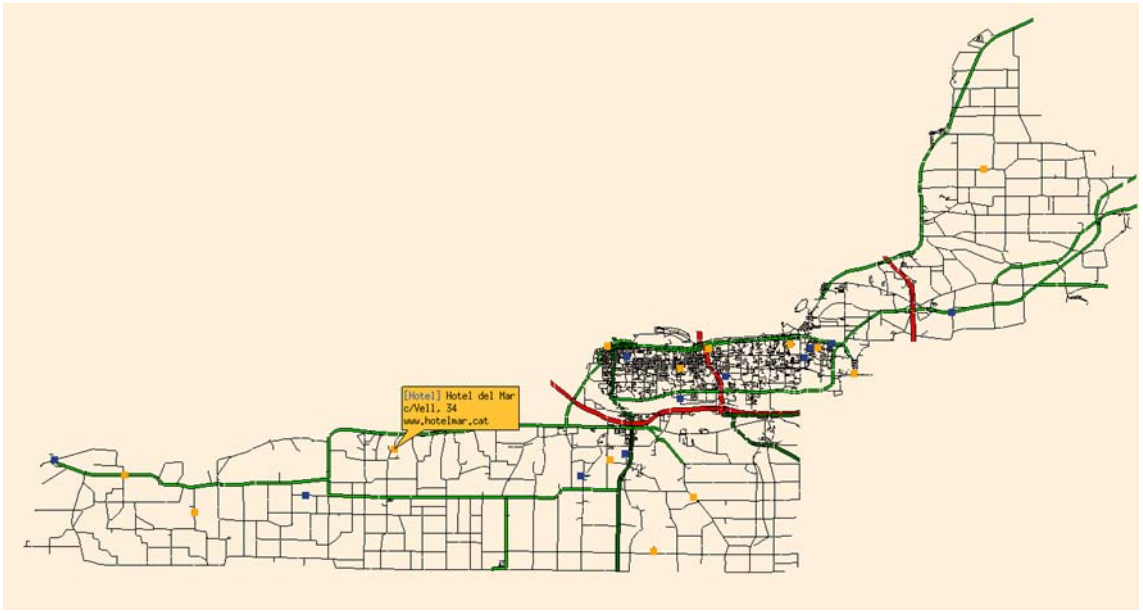


Figura 7.17: Problema Ak-NN amb k=1 i funció MinMax

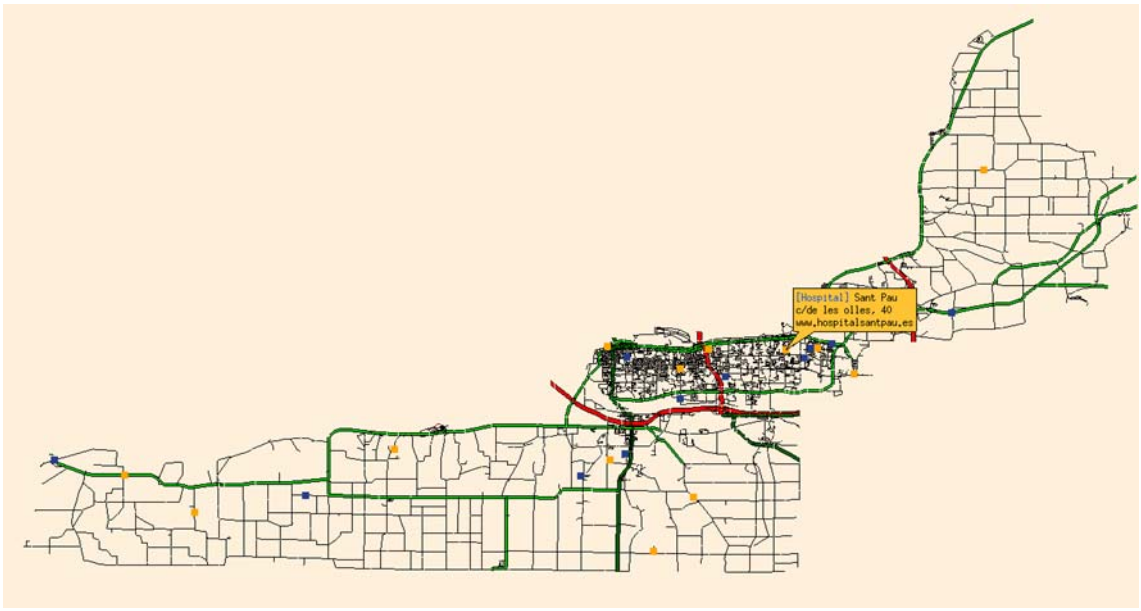


Figura 7.18: Problema Ak-NN amb k=1 i funció MinSum

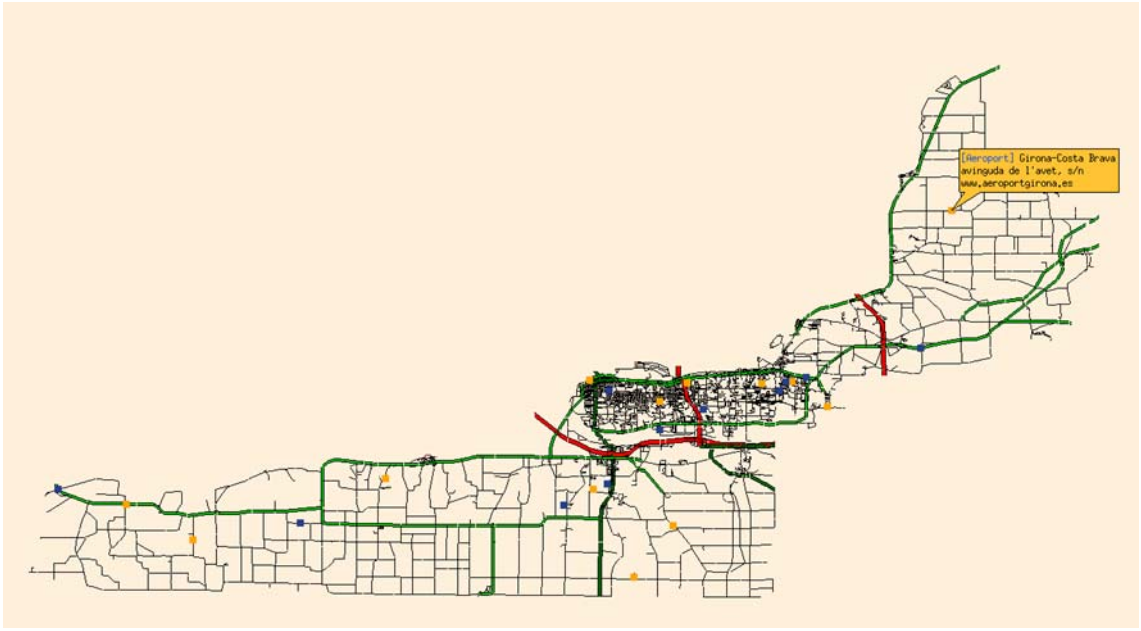


Figura 7.19: Problema Ak-NN amb k=1 i funció MaxMin

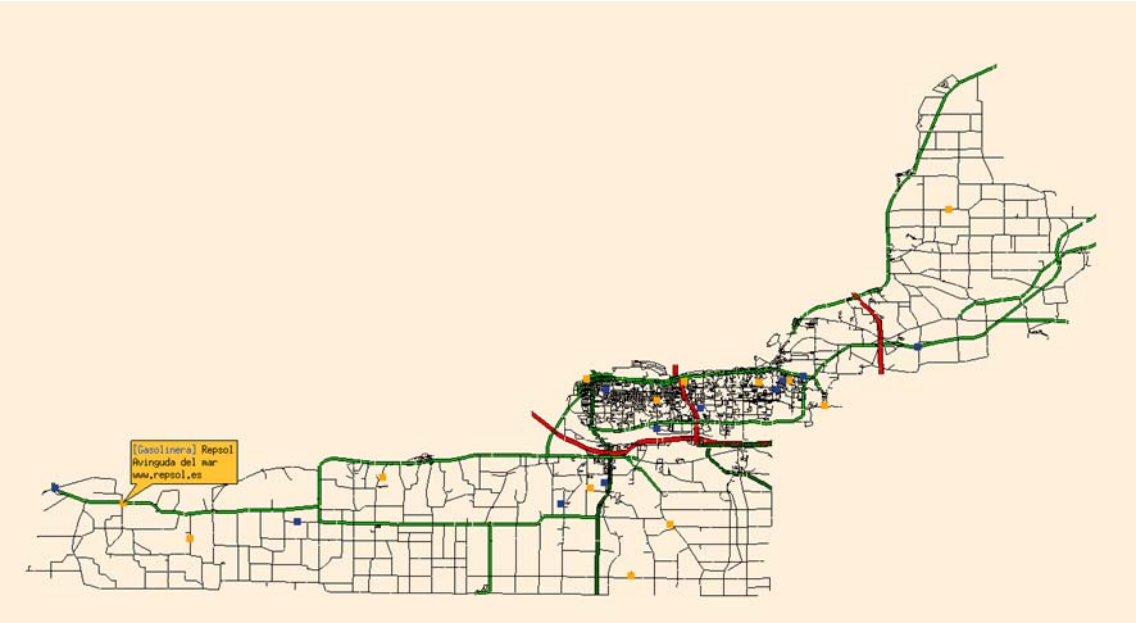


Figura 7.20: Problema Ak-NN amb k=1 i funció MaxSum

7.2. Temps de càlcul

En aquest apartat s'avalua el temps de càlcul de les principals funcionalitats de l'aplicació. Per al càlcul de cada un dels temps així com les imatges mostrades en aquesta memòria s'han extret amb un màquina Intel Core2 Duo a 2.40GHz amb 2GB de RAM i una targeta gràfica NVIDIA GeForce 8600 GT. Per a fer el càlcul de cada un dels temps s'ha fet la mitjana de varies execucions.

S'han utilitzat diferents xarxes de carreteres per dur a terme les execucions:

Xarxa	Nombre de vèrtexs	Nombre d'arestes
Rock Island	10.344	12.797
Washington	80.293	99.621
Nevada	278.962	336.634
Nova York	724.366	910.310

Figura 7.21: Taula de xarxes de carreteres utilitzades

7.2.1. Estructures de dades

Hem calculat el temps necessari per a crear el QuadTree comprimit de vèrtexs i per a crear l'RTree d'arestes. Podem veure que el temps de creació depèn del nombre d'elements que conté l'estructura de dades. Encara que es necessiti aquest temps per a generar l'estructura de dades, la cerca posterior de vèrtexs i arestes és logarítmica i evitem fer recorreguts a un vector, fet que seria molt més costós.

Xarxa	Creació QuadTree
Rock Island	0'027(s)
Washington	0'159(s)
Nevada	0'602(s)
Nova York	2'186(s)

Figura 7.22: Temps de càlcul QuadTree

Xarxa	Creació RTree
Rock Island	0'063(s)
Washington	0'400(s)
Nevada	1'495(s)
Nova York	3'715(s)

Figura 7.23: Temps de càlcul RTree

7.2.2. Parametrització de la xarxa

El càlcul de la parametrització de la xarxa depèn del nombre d'arestes que cal mapejar sobre la reixa rectangular. En la taula de la *figura 7.24* es pot veure el temps necessari per a calcular la parametrització en el diferents mapes. Cal remarcar que la mida de la reixa que s'utilitza és de 1280 x 881, fet que afectà alguns dels càlculs posteriors que es fan sobre la parametrització.

Xarxa	Parametrització
Rock Island	0'038(s)
Washington	0'042(s)
Nevada	0'117(s)

Figura 7.24: Temps de càlcul parametrització de la xarxa

7.2.3. Algorisme de Dijkstra i Camp Distàncies

El temps de càlcul de l'algorisme Dijkstra i de la funció camp de distàncies depèn de la mida del graf, tal i com es pot observar en la taula de la *figura 7.25*.

Xarxa	Dijkstra	Camp distàncies
Rock Island	0'006(s)	0'049(s)
Washington	0'055(s)	0'078(s)
Nevada	0'195(s)	0'162(s)

Figura 7.25: Temps de càlcul algorisme Dijkstra i Camp de distàncies

7.2.4. Diagrames de Voronoi

Si assumim que els camps de distàncies de cada una de les facilitats de la xarxa ja estan calculats, els temps de càlcul dels diferents diagrames de Voronoi són els que es detallen en la taula de la *figura 7.26*. Es pot observar que el temps de càlcul augmenta proporcionalment al augmentar el valor de k i al variar el nombre de facilitats de la xarxa.

Facilitats	Voronoi proper	Voronoi k=10	Voronoi k=25	Voronoi k=50	Voronoi k=100
25	0'436(s)	4'245(s)	-	-	-
50	0'782(s)	7'658(s)	18'835(s)	-	-
100	1'483(s)	14'596(s)	36'183(s)	72'198(s)	-
200	2'863(s)	28'505(s)	70'727(s)	138'994(s)	278'817(s)

Figura 7.26: Temps de càlcul diagrames de Voronoi

7.2.5. Problemes de proximitat

El temps de càlcul del problema k-NN es pot dir que depèn del temps de càlcul del diagrama de Voronoi d'ordre k. S'ha pogut comprovar que un cop generat el diagrama de Voronoi el temps de càlcul que cal afegir per a resoldre el problema k-NN és mínim i és el que es pot veure en la taula de la *figura 7.27*.

k=10	k=50
0'001(s)	0'0011(s)

Figura 7.27: Temps de càlcul problema k-NN

La resolució del problema Rk-NN es pot realitzar, com a màxim, amb 24 facilitats, per tant, no es poden obtenir massa resultats de càlcul de temps. En els

temps mostrats en la *figura 7.28* s'inclou el temps de calcular els camps de distàncies, el cercle(0'1145s) i els k-NN necessaris per a la resolució del problema. Es pot observar com el temps de càlcul augmenta si augmenta el valor de k. És degut al cost de calcular el diagrama de voronoi k dins la resolució dels k-NN.

Facilitats	k=1	k=10	k=20
24	5'470(s)	11'897(s)	18'961(s)

Figura 7.28: Temps de càlcul problema Rk-NN

En la taula de la figura 7.29 es poden veure els temps de càlcul del problema Ak-NN variant el nombre de facilitats i de punts de consulta. S'inclou en aquests temps, el temps de càlcul dels camps de distàncies de cada una de les facilitats utilitzades. Es pot observar com per a les funcions MinMax i MaxMin si no es varia el nombre de facilitats el temps de càlcul és constant. Per contra, en les funcions MinSum i MaxSum el temps de càlcul depèn del nombre de facilitats però també del nombre de punts de consulta de la xarxa. Al augmentar el nombre de punts de consulta augmenta el temps de càlcul degut al cost d'haver de llegir a cada pas de la suma el buffer de profunditat.

Facilitats	Punts de consulta	k=1 MinMax MaxMin	k=1 MinSum MaxSum
25	25	1'262(s)	1'352(s)
25	50	1'262(s)	1'465(s)
25	100	1'262(s)	1'553(s)
25	200	1'262(s)	1'801(s)
25	500	1'262(s)	1'316(s)
50	25	2'371(s)	2'444(s)
50	50	2'371(s)	2'495(s)
50	100	2'371(s)	2'625(s)
50	200	2'371(s)	2'885(s)
50	500	2'371(s)	3'694(s)
100	25	4'530(s)	4'621(s)
100	50	4'530(s)	4'650(s)
100	100	4'530(s)	4'775(s)
100	200	4'530	5'030
100	500	4'530	4'534

Figura 7.29: Temps de càlcul problema Ak-NN

8. Software

En aquest capítol s'esmenten les llibreries i software utilitats per tal de dur a terme aquest projecte. Es mostraran les eines que han permès la implementació de l'aplicació així com els programes necessaris per construir la documentació corresponent.

8.1. Llibreries utilitzades

8.1.1. Llibreries utilitzades per l'entorn gràfic

8.1.1.1. Llibreries bàsiques d'OpenGL:

OpenGL (Open Graphics Library) és una interfície de software per a gràfics de hardware. La interfície consisteix en un gran nombre de procediments i funcions que permeten al programador especificar els objectes i les operacions necessàries per produir imatges gràfiques de gran qualitat.

Normalment són utilitzades per a crear imatges en color que contenen objectes tridimensionals. Disposa d'un gran nombre de comandes que afecten a les operacions realitzades sobre gràfics de hardware. Qualsevol targeta gràfica amb accelerador conté implementada per hardware la llibreria GL. En el cas que no la contingui pot ser instal·lada per software a través de la llibreria MESA, tot i que en aquest cas les operacions aniran molt més lentes ja que no ho executa la CPU.

En la nostra aplicació les comandes d'OpenGL han estat utilitzades per representar les xarxes de carreteres.

Pàgina web: http://www.opengl.org

8.1.1.2. Llibreries Cg:

És un llenguatge compatible amb les dues API gràfiques més importants: OpenGL i DirectX. Ha estat desenvolupat per NVIDIA amb col·laboració amb Microsoft. És un llenguatge similar al llenguatge de programació C, amb la mateixa sintaxi per a les declaracions, les crides a funcions i la majoria dels tipus de dades. No obstant, té algunes millores i modificacions per a facilitar la codificació de programes que es compilin en codi assemblador de GPU optimitzat.

En la nostra aplicació s'utilitza per aprofitar la potència de la GPU i accelerar els càlculs amb la finalitat de resoldre els problemes més ràpidament.

Pàgina web: http://www.nvidia.com

8.1.1.3. Llibreries aportades per les Qt:

Qt és un toolkit (joc d'eines) pel desenvolupament d'aplicacions en mode gràfic que utilitza el llenguatge de programació C++. Ha estat creada i mantinguda per Trolltech, una companyia de software de nivell internacional. No és tan sols una llibreria de classes. Es tracta d'una multiplataforma que permet crear interfícies gràfiques d'usuari (GUI) amb facilitat, ja que disposa d'una excel·lent documentació.

A més a més, Qt és suportada per un gran nombre de sistemes operatius com MS/Windows, Unix/X11, Macintosh i Embedded. La seva gran portabilitat no serveix com a inconvenient pel programador ja que el codi utilitzat en qualsevol de les plataformes és exactament el mateix. Per aquest motiu, tan sols cal programar-ho una única vegada i llavors compilar-lo segons el tipus de variant de Qt escollida.

A partir d'un conjunt de classes aportades per Qt s'ha pogut realitzar la interfície gràfica de l'aplicació. Aquestes han permès dissenyar la finestra del programa amb cadascun dels diferents tipus de controls que hi apareixen. Utilitzant la programació amb events que Qt permet fer servir s'ha facilitat la tasca d'interacció amb l'usuari.

Pàgina web: http://www.trolltech.com

8.2. Software utilitzat

8.2.1. Software utilitzat per a la implementació

8.2.1.1. KDevelop:

KDevelop és una eina que va ser creada amb l'objectiu de desenvolupar un IDE (Entorn de desenvolupament integrat) fàcil d'utilitzar per KDE. Des dels seus inicis, l'IDE KDevelop està públicament disponible sota la GPL. La GPL és una llicència creada per *Free Software Fundation* orientada principalment als temes de distribució, modificació i ús del software. El seu propòsit és declarar que el software d'aquesta llicència és software lliure. A més a més, el Kdevelop suporta molts llenguatges de programació.

Aquesta eina ha estat utilitzada per implementar tot el codi de la nostra aplicació ja que és un bon suport per el llenguatge C++.

Pàgina web: http://www.kdevelop.org/

8.2.1.2. Ruby:

Ruby és un llenguatge de programació dinàmic, flexible i de codi obert enfocat a la simplicitat i productivitat.

El Ruby ens ha permès generar un fitxer de xarxa de carreteres a partir d'un fitxer TIGER/Lines adaptat a les necessitats de l'aplicació.

Pàgina web: http://www.ruby-lang.org

8.2.2. Software utilitzat per la documentació

8.2.2.1. Microsoft Office Word 2007:

Aquesta eina és el processador de textos del Microsoft Office 2007. Word 2007 és la versió més recent del processador de textos més popular sobre el sistema de Windows.

Aquest programa ha estat utilitzat per la creació de tota la memòria del projecte.

Pàgina web: *http://www.microsoft.com*

8.2.2.2. Microsoft Visio Professional 2003:

Es tracta d'un programa que permet crear tot tipus de diagrames orientats al disseny i l'anàlisi del programari. Des del punt de vista empresarial dona la facilitat de documentar, dissenyar i analitzar mitjançant una sèrie de plantilles i formes que admeten projectes d'administració de processos empresarials.

Aquest software permet crear diagrames per tal de descriure els conceptes més ràpidament i comunicar d'aquesta manera informació amb major eficàcia.

Aquest programa ha estat utilitzat durant el procés d'anàlisi per crear els diagrames UML que es mostren en la memòria.

Pàgina web: *http://www.microsoft.com*

9. Conclusions

9.1. Conclusions

L'objectiu principal d'aquest projecte era la realització d'una aplicació que permetés visualitzar xarxes de carreteres, mantenir-hi objectes i resoldre problemes de proximitat mitjançant una interfície gràfica senzilla i intuïtiva.

Aquest objectiu, com s'ha pogut comprovar, ha estat complert de forma satisfactòria. Tot el conjunt de requeriments definits des d'un principi han estat assolits en l'aplicació final.

Per tal d'aconseguir l'objectiu principal descrit ha calgut dissenyar una interfície gràfica d'usuari còmoda i funcional per tal de poder realitzar cadascuna de les funcionalitats de l'aplicació.

S'han buscat diferents fitxers de text que contenen xarxes de carreteres i s'han adaptat a les necessitats de la nostra aplicació. A partir d'aquest fitxer s'ha creat el graf, que representa una xarxa.

En aquesta xarxa, s'ha fet la representació d'objectes estàtics i objectes mòbils a través d'una estructura de dades que permet crear-los i eliminar-los. Així mateix, s'han creat altres estructures de dades per a distribuir els vèrtexs i les arestes del graf.

S'ha implementat l'algorisme de camins mínims de Dijkstra, la funció camp de distàncies, el cercle i diferents diagrames de Voronoi amb la finalitat de poder resoldre els problemes de proximitat k-veïns més propers, k-veïns més propers inversos i k-veïns més propers agregats sobre la xarxa de carreteres. A més, s'ha implementat un algorisme que permet calcular quatre punts d'interès.

Gran part d'aquests càlculs han estat possibles gràcies a la parametrització que s'ha fet de la xarxa de carreteres amb al qual s'ha pogut compactar les arestes del graf i facilitar els càlculs.

Utilitzant la potència de la GPU s'han pogut accelerar els càlculs i, per tant, obtenir millors temps de resposta.

No només s'han complert els requisits sinó que s'han superat. S'ha enriquit la interfície gràfica millorant la interacció amb l'usuari i s'han desenvolupat noves funcionalitats a l'aplicació.

Cal dir que per poder obtenir l'aplicació resultant s'han hagut d'aprendre diferents conceptes que per a nosaltres eren desconeguts: els conceptes teòrics explicats a l'estudi previ, el funcionament de les QT4 i la programació de la GPU. A més, amb la realització d'aquest projecte hem pogut aprofundir els nostres coneixements sobre les llibreries d'OpenGL.

Durant el transcurs del procés de realització de l'aplicació han sorgint diversos problemes que, amb més o menys temps s'han aconseguit resoldre.

La realització d'un projecte en grup ha estat molt profitosa ja que hem millorat el treball en grup i hem après a organitzar-nos millor. Tot i que cada membre del grup ha desenvolupat una part de les funcionalitats, s'ha creat una sola aplicació conjunta més extensa on s'hi reflecteix tot el treball realitzat.

Una de les dificultats més importants que hem trobat treballant en grup és portar un control de versions de la implementació. Ha estat de gran importància realitzar una bona planificació de tasques i respectar-les estrictament per no haver de modificar el treball de l'altre membre del grup.

9.2. Temporització

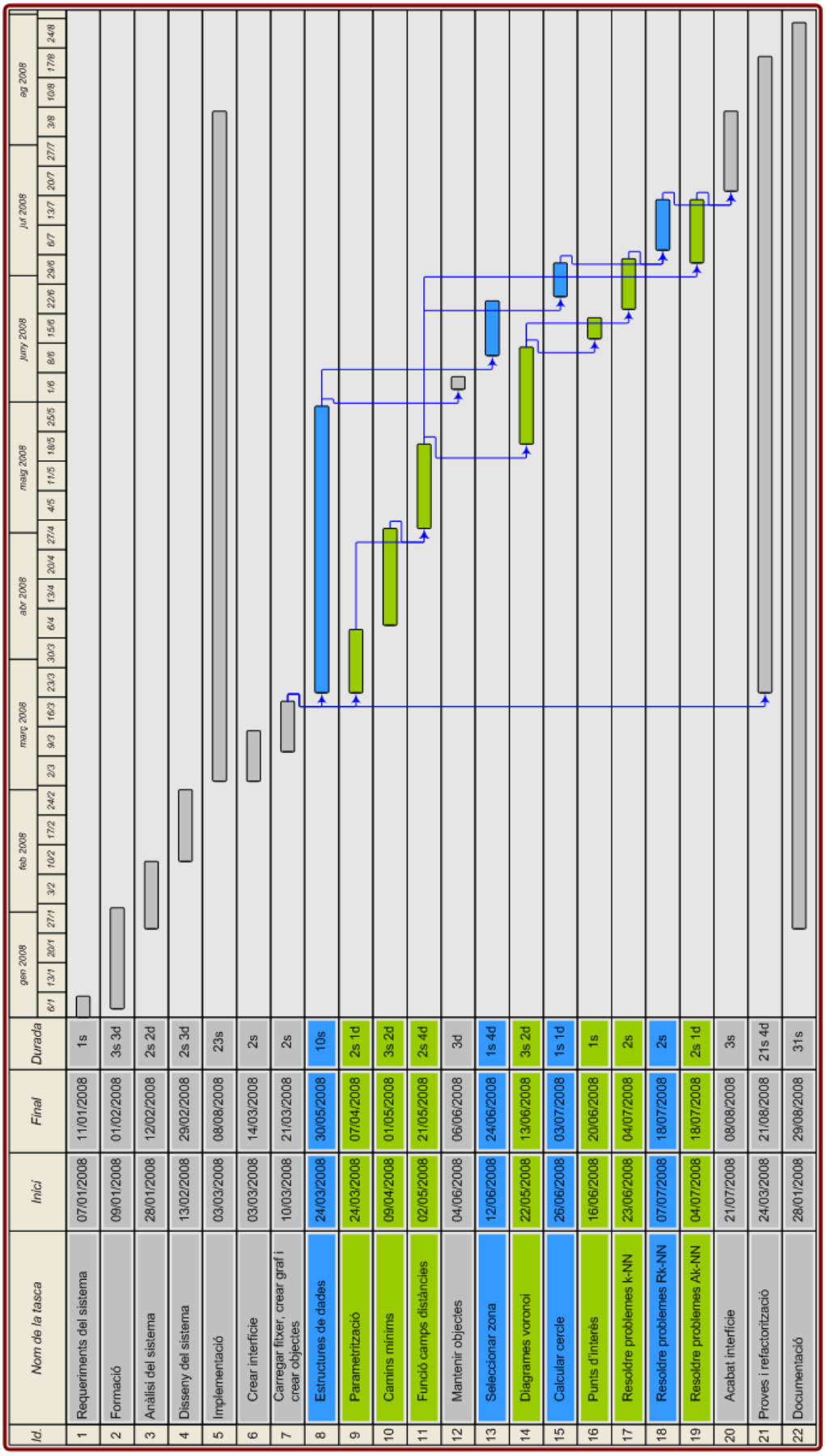
En aquest apartat s'explicarà la planificació realitzada quan es va iniciar el projecte i com s'ha anat modificant a mesura que s'ha anat desenvolupant.

Quan es va començar el projecte es va creure que era molt important realitzar una bona planificació entre els membres del grup. Com s'ha pogut observar en els diagrames de casos d'ús, hi ha funcionalitats que depenen d'altres i, per això, calia distribuir bé el temps perquè en cap moment quedés un dels membres sense feina.

Cal dir que la planificació inicial s'ha anat modificant a mesura que s'avançava el projecte, ja que en un principi era gairebé impossible preveure el temps que requeriria la realització de cada tasca. Tot i això, el resultat ha estat del tot satisfactori perquè s'ha aconseguit mantenir un bon ritme de treball.

Un dels canvis més importants que s'ha fet a la temporització inicial és posposar la data de finalització. En un principi molt optimista es va decidir marcar el final del projecte al juny, però va haver-hi algun. El llarg temps que es va necessitar per a la recerca, l'estudi previ i la formació dels membres del grup, que va fer atraçar totes les altres tasques. Més endavant, veient que s'acostava la data de finalització i s'anava just de temps, es va decidir modificar tota la planificació i desenvolupar el projecte fins al setembre. Aquesta decisió va ser presa per tal de poder acabar-lo amb tranquil·litat i poder dedicar més temps a realitzar proves, corregir possibles errors i afegir a l'aplicació noves funcionalitats, poc importants per a resoldre problemes de proximitat, però que serveixen per a enriquir l'aplicació.

A la següent figura es pot observar com ha quedat finalment la temporització d'aquest projecte un cop realitzats tots els canvis esmentats.



Id.	Nom de la tasca	Inici	Final	Durada
1	Requeriments del sistema	07/01/2008	11/01/2008	1s
2	Formació	09/01/2008	01/02/2008	3s 3d
3	Anàlisi del sistema	28/01/2008	12/02/2008	2s 2d
4	Disseny del sistema	13/02/2008	29/02/2008	2s 3d
5	Implementació	03/03/2008	08/08/2008	23s
6	Crear interfície	03/03/2008	14/03/2008	2s
7	Carregar fitxer, crear graf i crear objectes	10/03/2008	21/03/2008	2s
8	Estructures de dades	24/03/2008	30/05/2008	10s
9	Parametrització	24/03/2008	07/04/2008	2s 1d
10	Camins mínims	09/04/2008	01/05/2008	3s 2d
11	Funció camps distàncies	02/05/2008	21/05/2008	2s 4d
12	Mantenir objectes	04/06/2008	06/06/2008	3d
13	Seleccionar zona	12/06/2008	24/06/2008	1s 4d
14	Diagrames voronoi	22/05/2008	13/06/2008	3s 2d
15	Calcular cercle	26/06/2008	03/07/2008	1s 1d
16	Punts d'interès	16/06/2008	20/06/2008	1s
17	Resoldre problemes k-NN	23/06/2008	04/07/2008	2s
18	Resoldre problemes Rk-NN	07/07/2008	18/07/2008	2s
19	Resoldre problemes Ak-NN	04/07/2008	18/07/2008	2s 1d
20	Acabat interfície	21/07/2008	08/08/2008	3s
21	Proves i refactorització	24/03/2008	21/08/2008	21s 4d
22	Documentació	28/01/2008	29/08/2008	31s

Figura 9.1: Temporització

9.3. Ampliacions

En aquesta aplicació s'han implementat els objectes mòbils sobre la xarxa de carreteres, però no s'han tingut en compte per a realitzar els càlculs. Totes les funcionalitats desenvolupades són vàlides per a aquest tipus d'objectes i es podria fer l'adaptació fàcilment, tot i que caldria examinar amb profunditat com visualitzar els resultats en temps real.

10. Manual d'usuari

Gran part d'aquest projecte ha consistit en la creació d'una interfície gràfica que permetés interactuar l'usuari amb l'aplicació. Aquesta interfície ens permet accedir a totes aquelles funcionalitats comentades en els apartats de requeriments i anàlisi.

L'aplicació és, per tant, un visualitzador de xarxes de carreteres que permet afegir i esborrar objectes amb la finalitat de poder resoldre problemes de proximitat.

En aquest capítol s'explicarà el funcionament d'aquesta interfície per tal de conèixer totes les opcions de què disposa i en veurem exemples.

10.1. Inici de l'aplicació

Quan s'inicia l'aplicació apareix una finestra amb el menú principal i una barra d'eines. La majoria d'aquestes opcions estan inhabilitades perquè no té cap sentit utilitzar-les sense que hi hagi cap xarxa de carreteres carregada.

La part central de l'aplicació, i la més gran, l'ocupa una zona grisa que representa l'escriptori. Inicialment aquest està buit.

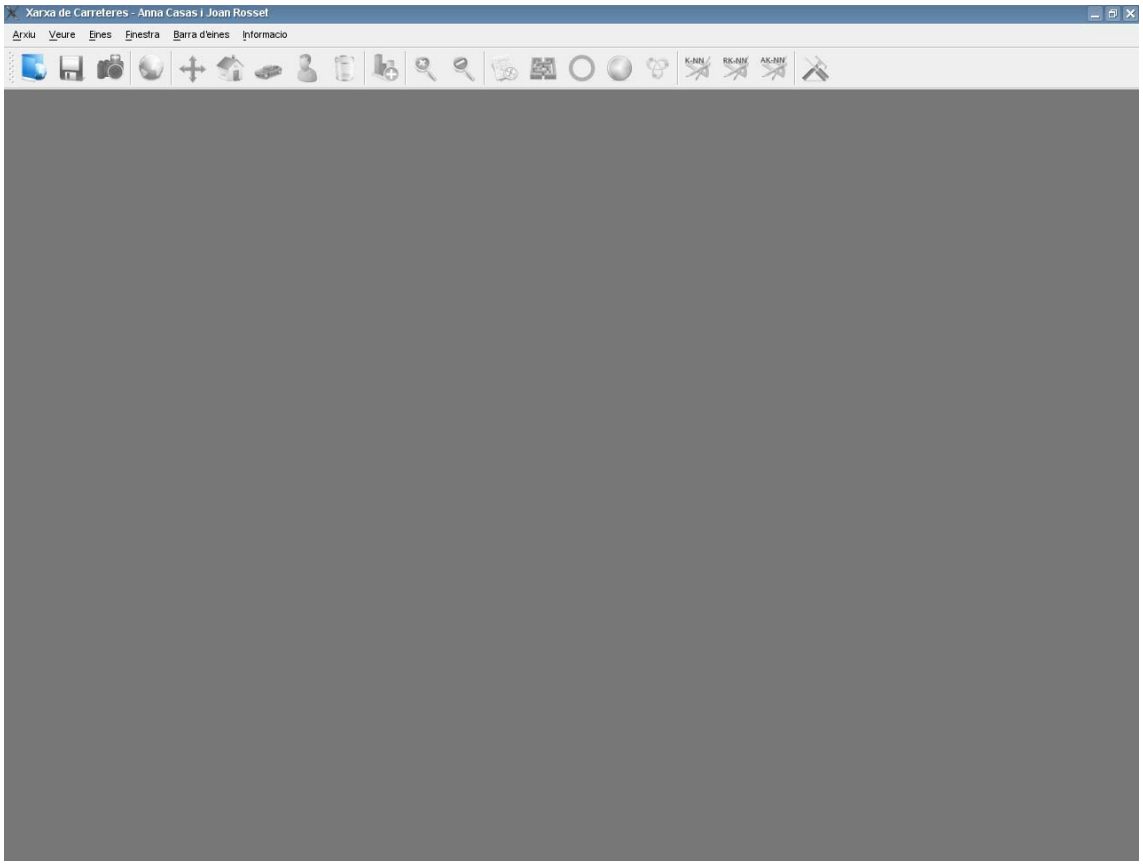


Figura 10.1: Aspecte inicial de l'aplicació

10.1.1. Menú principal

El menú principal (*figura 10.2*) es troba situat a la part superior de l'aplicació.

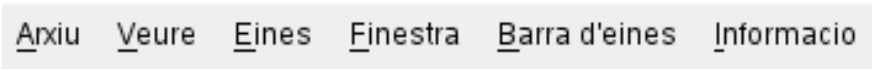


Figura 10.2: Menú principal

Cadascun dels ítems del menú principal conté un submenú amb diferents opcions. Des d'aquestes opcions es poden accedir a totes les funcionalitats de l'aplicació.

10.1.2. Barra d'eines

La barra d'eines (*figura 10.3*) es troba situada sota el menú principal.

Figura 10.3: Barra d'eines

Aquesta barra conté un conjunt d'ítems que permet accedir a les opcions més importants de l'aplicació. A totes aquestes accions també s'hi pot accedir utilitzant el menú principal. A la següent taula es pot veure quina acció realitza cada ítem i el seu equivalent utilitzant el menú principal:

Icona	Acció	Ubicació al menú principal
	Carregar fitxer	Menú arxiu/Obrir...
	Guardar objectes	Menú arxiu/Guardar...
	Capturar pantalla	Menú arxiu/Guardar imatge...
	Veure xarxa carreteres	Menú eines/Veure mapa
	Mode visualitzar	Menú eines/Modes/Visualitzar
	Mode afegir facilitat	Menú eines/Modes/Afegir facilitat

	Mode afegir car	Menú eines/Modes/Afegir car
	Mode afegir punt de consulta	Menú eines/Modes/Afegir punt de consulta
	Mode eliminar objecte	Menú eines/Modes/Eliminar objecte
	Afegir objectes	Menú eines/Afegir objectes
	Seleccionar zona	Menú eines/Seleccionar zona
	Desseleccionar zona	Menú eines/Tornar al mapa original
	Camins mínims Dijkstra	Menú eines/Dijkstra
	Funció camps distàncies	Menú eines/Camps distàncies
	Calcular cercle	Menú eines/Marcar cercle
	Calcular Voronoi	Menú eines/Voronoi
	Calcular punts d'interès	Menú eines/Punts d'interès
	Resoldre problema k-NN	Menú eines/K-NN
	Resoldre problema invers k-NN	Menú eines/RK-NN

	Resoldre problema agregat k-NN	Menú eines/AK-NN
	Preferències de l'aplicació	Menú eines/Preferències

Figura 10.4: Barra d'eines: Icones

10.1.3. Escriptori

Aquesta part de la interfície és la més gran i és la que s'encarrega de mostrar totes les finestres que hi ha obertes a l'aplicació. Dins l'escriptori les finestres es poden desplaçar, minimitzar, maximitzar i ocultar.

10.2. Menú: Arxiu

El menú arxiu, tal i com es mostra a la *figura*, té els següents ítems:

Figura 10.5: Menú: Arxiu

10.2.1. Menú Arxiu: Obrir…

Aquesta opció del menú permet carregar una xarxa de carreteres a partir d'un fitxer de text. Aquest fitxer ha de tenir la extensió ".map". Quan ha carregat el fitxer de text automàticament busca si hi ha un fitxer d'objectes que tingui el mateix nom però amb la extensió ".map_data". En cas de trobar-lo preguntarà a l'usuari si vol carregar els objectes.

A continuació es pot veure un exemple de com carregar una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.6)* que permet navegar per a seleccionar un fitxer amb extensió ".map". Seleccionem el mapa que volem carregar.

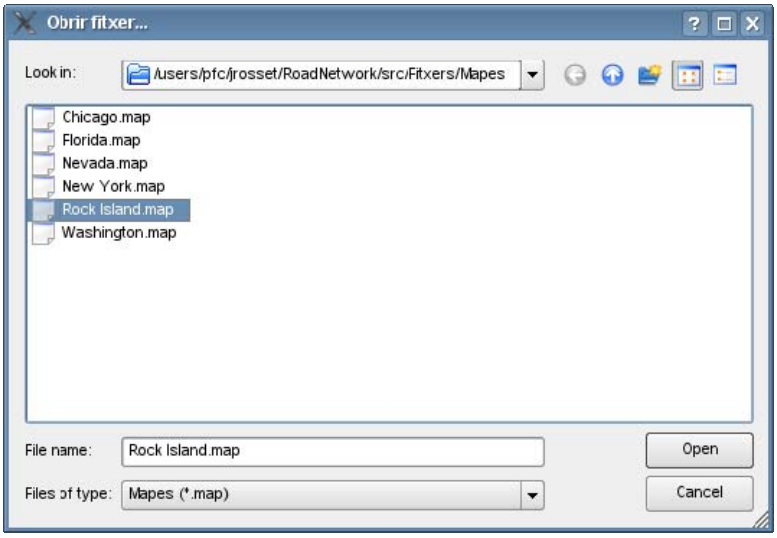


Figura 10.6: Finestra per seleccionar un fitxer

- Ens apareix una barra de progrés *(figura 10.7)* que ens indica en tot moment la acció que s'està realitzant i gràcies al tant per cent ens podem fer una idea del temps que falta per a tenir la xarxa de carreteres carregada.

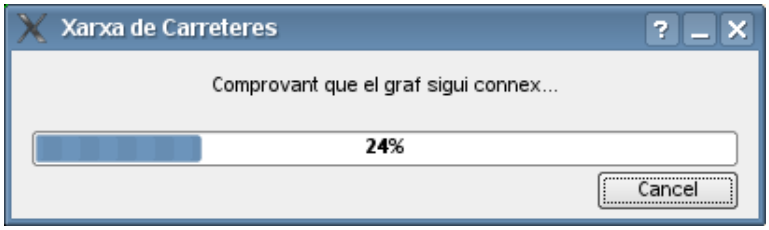


Figura 10.7: Finestra de progrés de carregar un fitxer

- Si ha trobat un fitxer d'objectes, ens apareix un missatge *(figura 10.8)* que ens demana si el volem carregar.

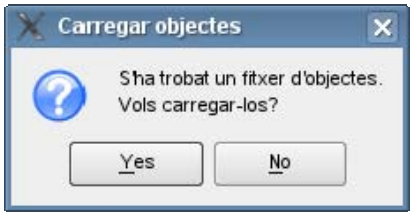


Figura 10.8: Seleccionar si carregar el fitxer d'objectes

- El mapa s'ha carregat correctament i ja el podem visualitzar a la finestra *(figura 10.9)*. Podem veure com algunes opcions del menú principal i de la barra d'eines s'han habilitat.

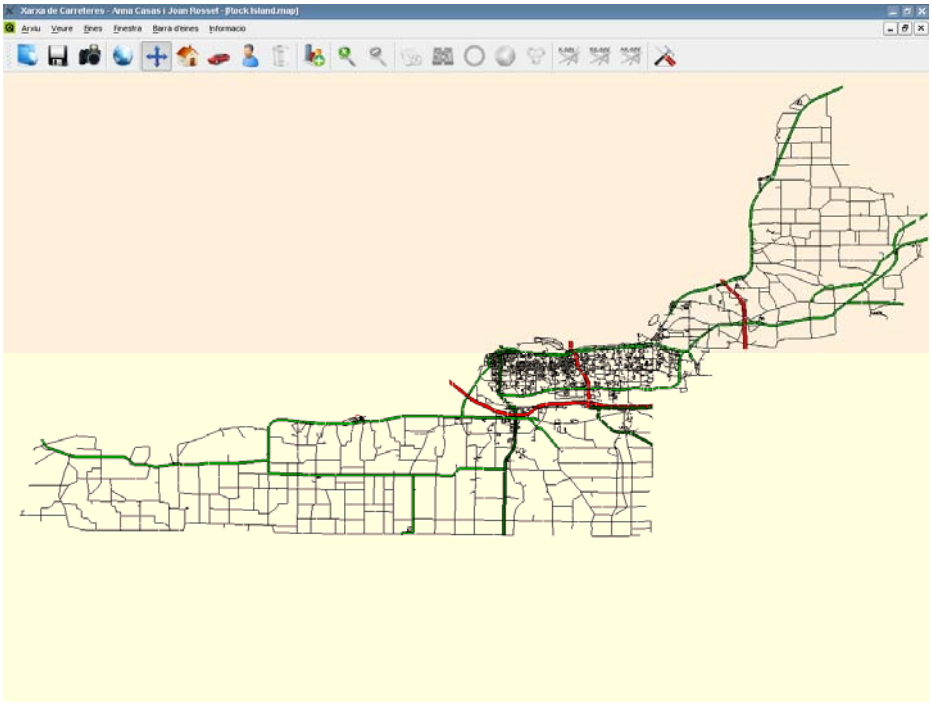


Figura 10.9: Visualització de la xarxa de carreteres

- En qualsevol moment també podem veure la finestra on es mostra la parametrització *(figura 10.10)* de la xarxa de carreteres. Per a fer-ho hi anirem pel menú finestra i seleccionarem la finestra parametrització.

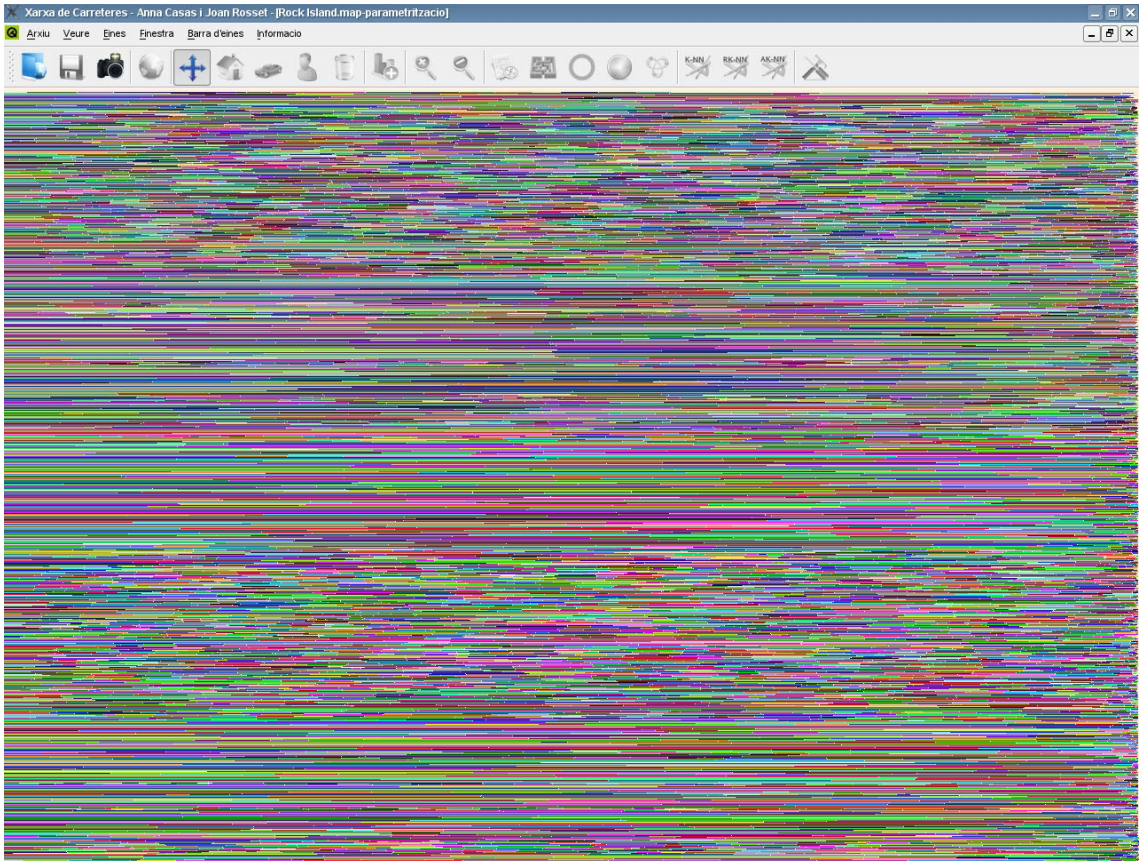


Figura 10.10: Visualització de la parametrització de la xarxa de carreteres

10.2.2. Menú Arxiu: Guardar…

Aquesta opció del menú permet guardar tots els objectes que hi ha a la xarxa de carreteres en un moment determinat dins d'un fitxer de text. El nom del fitxer serà el mateix que el nom del fitxer de la xarxa de carreteres canviant-li la extensió ".map" per ".map_data".

A continuació es pot veure un exemple de com guardar els objectes d'una xarxa de carreteres mitjançant aquesta opció:

- Inicialment tenim una xarxa de carreteres amb més d'un objecte *(figura 10.11)*.

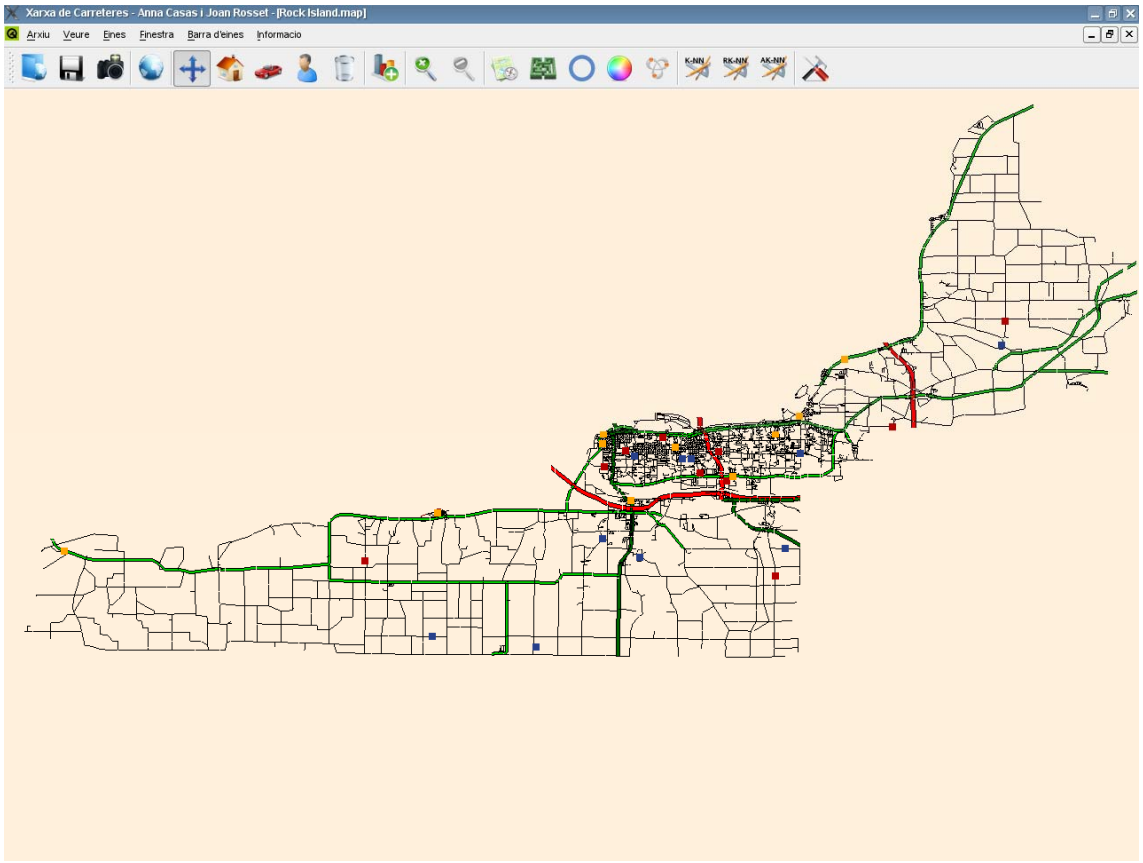


Figura 10.11: Xarxa de carreteres amb objectes

- Premem la icona i automàticament es guarden els objectes dins el fitxer de text. Si tot ha anat bé, ens apareix una finestra confirmant-ho.

Figura 10.12: Finestra confirmació objectes guardats

10.2.3. Menú Arxiu: Guardar imatge…

Aquesta opció del menú permet realitzar una captura de pantalla de la finestra que estem visualitzant en un moment determinat. Aquesta captura es realitza amb el mateix punt de vista i zoom que es té seleccionat en el moment de fer-la.

A continuació es pot veure un exemple de com guardar una imatage d'una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.13)* que permet navegar per a seleccionar una carpeta i introduir el nom del fitxer que volem guardar. Es podrà escollir guardar-la en format ".jpg" o ".bmp". Si no es posa cap extensió, per defecte l'aplicació agafa ".jpg".

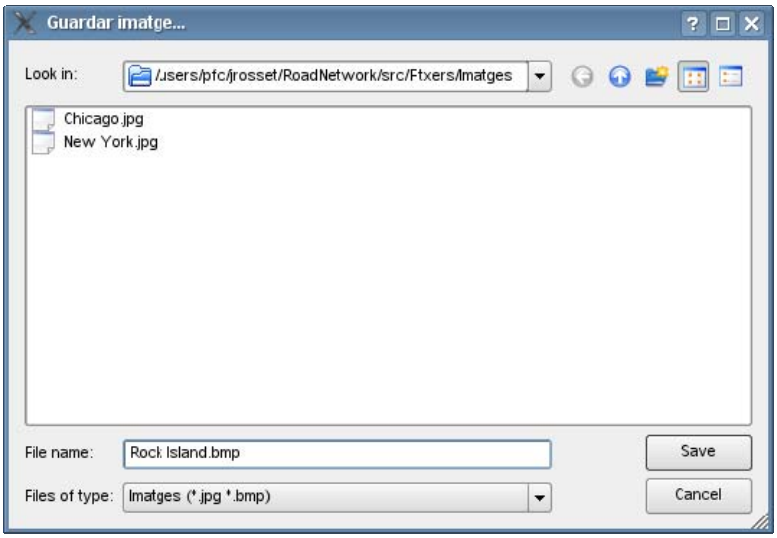


Figura 10.13: Finestra per introduir un fitxer

10.2.4. Menú Arxiu: Sortir

Aquesta opció del menú permet abandonar l'aplicació. Quan aquesta es tanqui, immediatament s'eliminarà tot el que teníem representat.

10.3. Menú: Veure

El menú Veure s'encarrega de mostrar o ocultar els objectes de la xarxa de carreteres. Com es pot veure a la *figura,* té els següents ítems:

Figura 10.14: Menú: Veure

- **Vèrtexs:** Quan la opció està activada, els vèrtexs del graf es mostren. En cas contrari, els vèrtexs quedaran ocults. Per defecte, aquests no es mostraran ja que la xarxa de carreteres es visualitza millor si només hi ha les arestes(carreteres).

- **Arestes:** Igual que en el cas anterior, aquest ítem indica si s'han de mostrar les arestes o ocultar-les.

- **Facilitats:** Ítem per a mostrar o ocultar les facilitats.

- **Punts de consulta:** Ítem per a mostrar o ocultar els punts de consulta.

- **Vehicles:** Ítem per a mostrar o ocultar els vehicles.

- **Moviment dels vehicles:** Aquesta opció indica si es vol visualitzar el moviment dels vehicles a través de la carretera. Si es desactiva aquesta opció, els vehicles es visualitzaran aturats, però internament seguiran movent-se. Aquest ítem és molt útil ja que mostrar aquest constant moviment requereix recursos i d'aquesta manera es pot evitar.

- **Engegar/Aturar vehicles:** Aquest ítem és semblant a l'anterior, amb la diferència que en aquest cas els vehicles s'aturen (inclús internament). Amb aquesta opció desactivada els recursos necessaris encara són menors.

- **QuadTree de vèrtexs:** Ítem per a mostrar o ocultar el QuadTree de vèrtexs. Aquesta opció serveix per a poder visualitzar sobre la xarxa de carreteres aquesta estructura de dades. L'únic sentit que té aquesta funcionalitat és poder visualitzar com s'ha creat l'estructura de dades.

- **QuadTree comprimit d'objectes:** Ítem per a mostrar o ocultar el QuadTree d'objectes. Igual que en el cas anterior, l'usuari pot comprovar com s'ha creat aquest QuadTree.

- **RTree:** Ítem per a mostrar o ocultar l'RTree d'arestes.

A continuació es pot veure un exemple *(figura 10.15)* de com queda una xarxa de carreteres visualitzant els vèrtexs i ocultant les arestes:

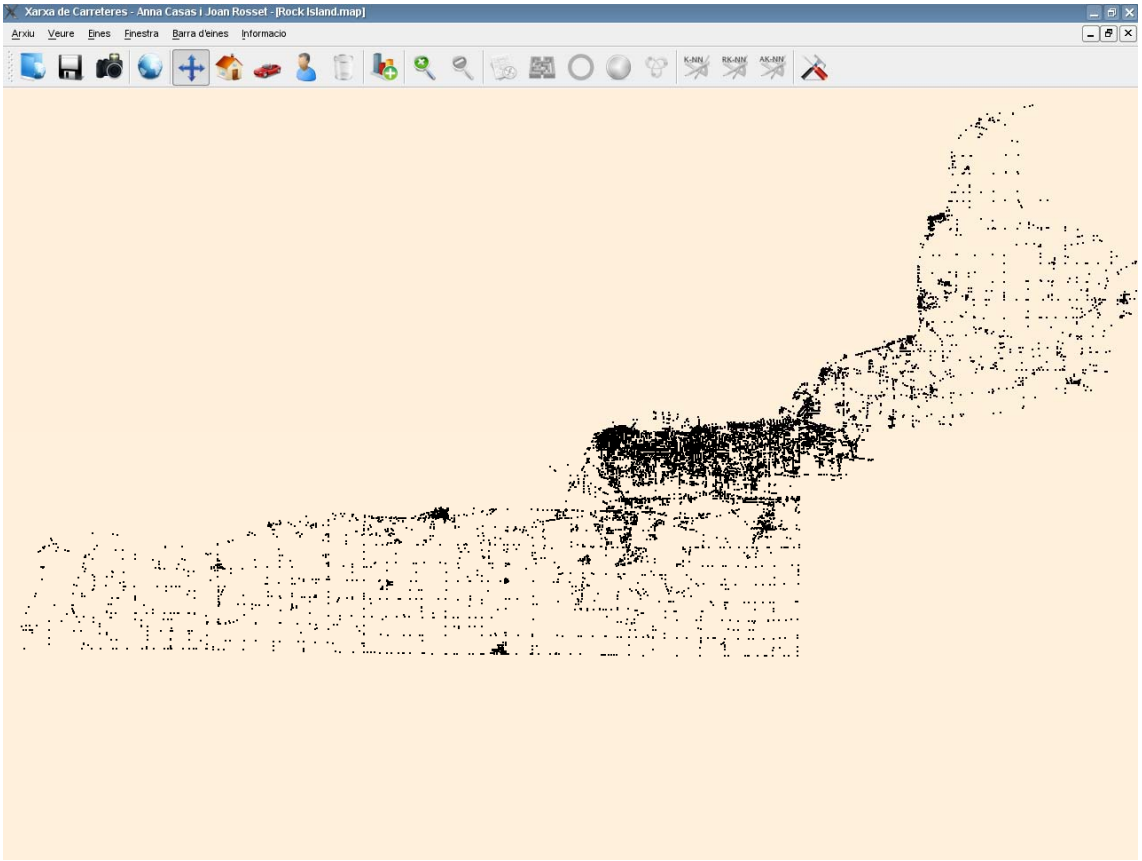


Figura 10.15: Xarxa de carreteres visualitzada amb vèrtexs

10.4. Menú: Eines

El menú Eines és el que conté la major part de les funcionalitats de l'aplicació. A la *figura* les podem observar:

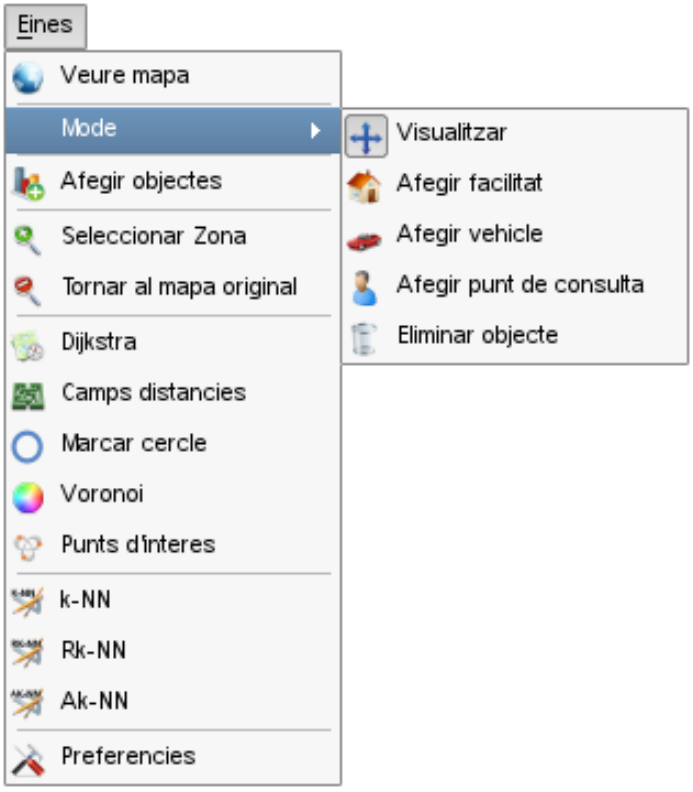


Figura 10.16: Menú: Eines

10.4.1. Menú Eines: Veure mapa

Aquesta opció permet visualitzar la xarxa de carreteres i els objectes sense mostrar cap altra cosa. S'utilitza quan s'ha resolt un problema i s'estan veient els resultats per pantalla. Prement aquesta icona, es reinicialitza el mapa i es deixa a punt per a tornar a calcular un altre problema.

10.4.2. Menú Eines: Mode

Aquesta opció del menú permet canviar el mode que es troba l'aplicació. A continuació s'explica com s'utilitza cadascun dels modes possibles.

10.4.2.1. Visualitzar

Aquest mode permet realitzar desplaçaments i zooms a la xarxa de carreteres mitjançant el ratolí d'una manera molt senzilla i intuïtiva.

Per a canviar el punt de vista, s'ha de prémer el botó esquerre del ratolí damunt de la xarxa de carreteres i, sense deixar-lo anar, anar-se desplaçant per la pantalla. Durant aquest desplaçament la xarxa de carreteres es va movent en la mateixa direcció que es mou el ratolí.

Per a canviar el zoom el procediment és molt semblant. El que cal fer és prémer el botó dret del ratolí sobre la xarxa de carreteres i, desplaçant-lo verticalment (eix y), es va canviant el zoom. Si el que es vol fer és aproximar-se a la xarxa de carreteres caldrà moure el ratolí cap amunt amb el botó dret premut. Per altra banda, si el que volem és allunyar-nos, caldrà moure el ratolí cap avall.

A continuació es pot veure un exemple d'un canvi de punt de vista i un zoom aplicat a una xarxa de carreteres.

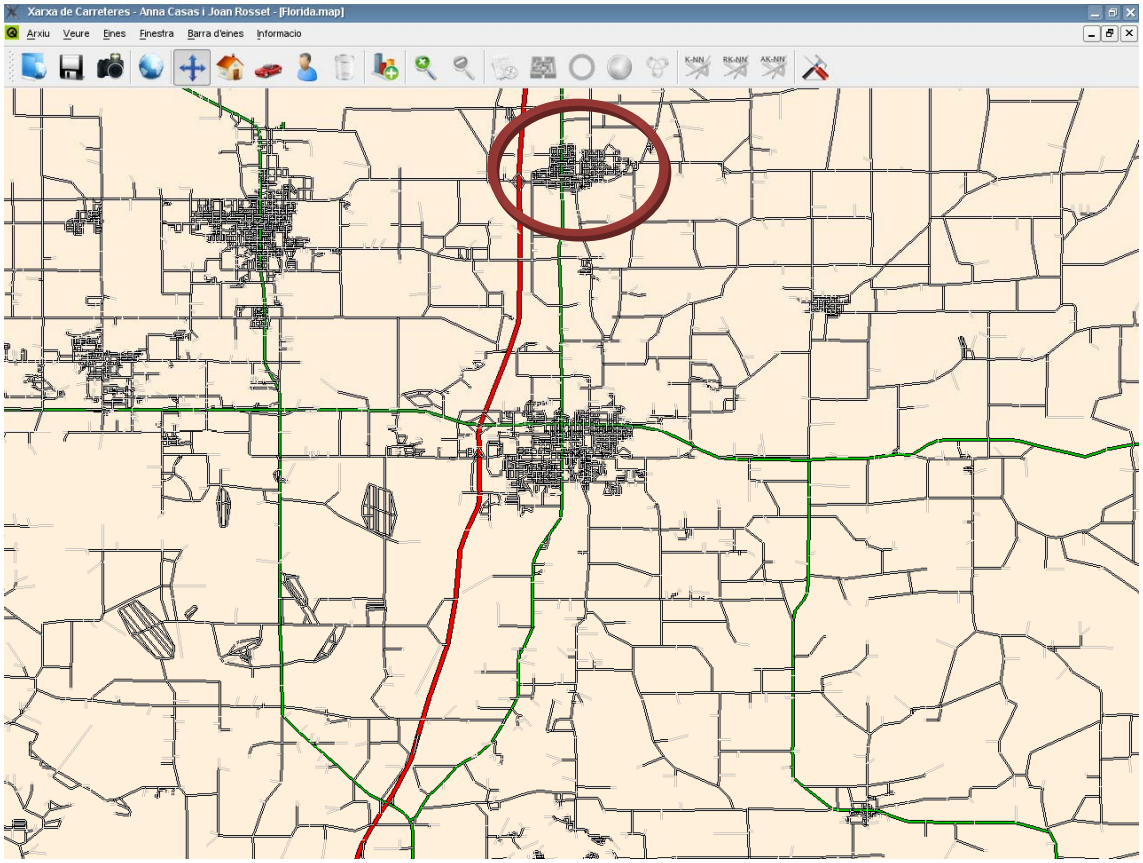


Figura 10.17: Xarxa de carreteres

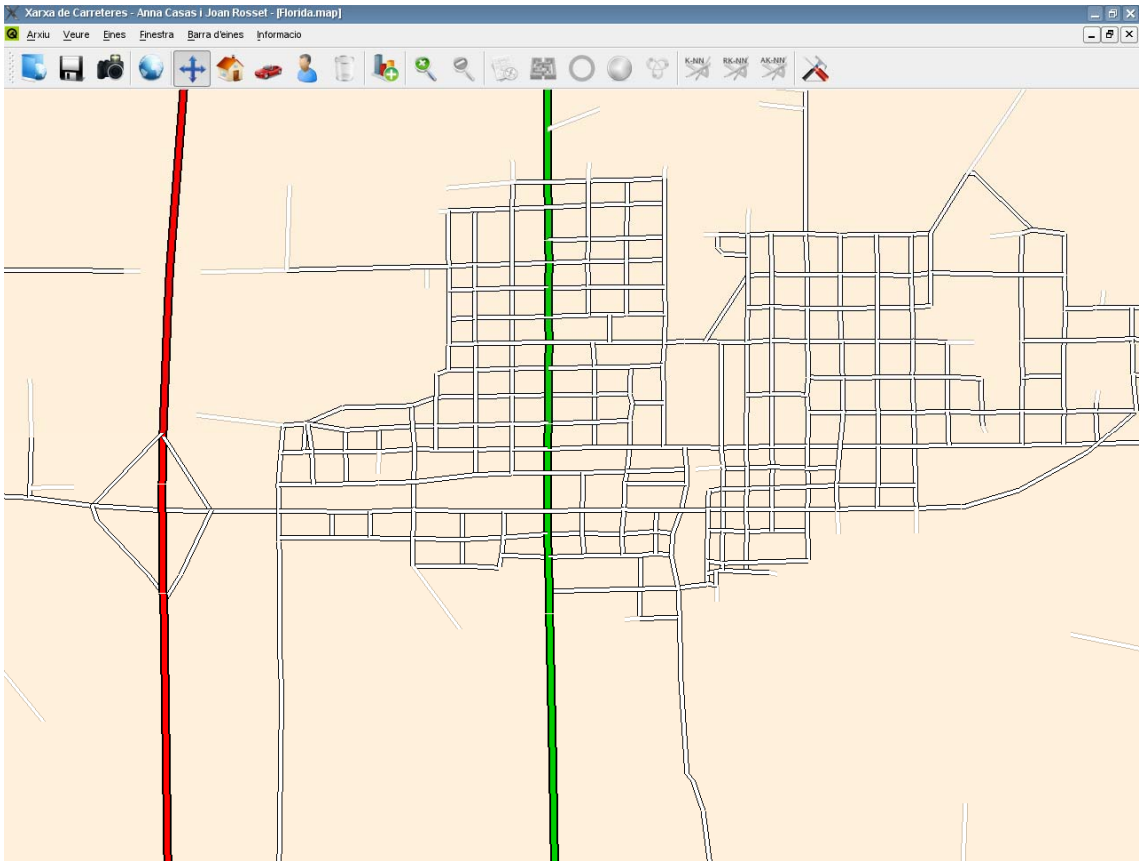


Figura 10.18: Xarxa de carreteres amb desplaçament i zoom

10.4.2.2. Afegir facilitat

Aquest mode permet afegir una facilitat a la xarxa de carreteres seleccionant la seva ubicació i introduint les seves dades.

A continuació es pot veure un exemple de com afegir una facilitat a una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i l'aplicació es posa automàticament en el mode d'afegir una facilitat. Ara, desplaçant el ratolí per sobre la xarxa de carreteres, ens apareix un punt virtual que ens indica a on s'afegiria si preméssim el botó esquerre del ratolí.

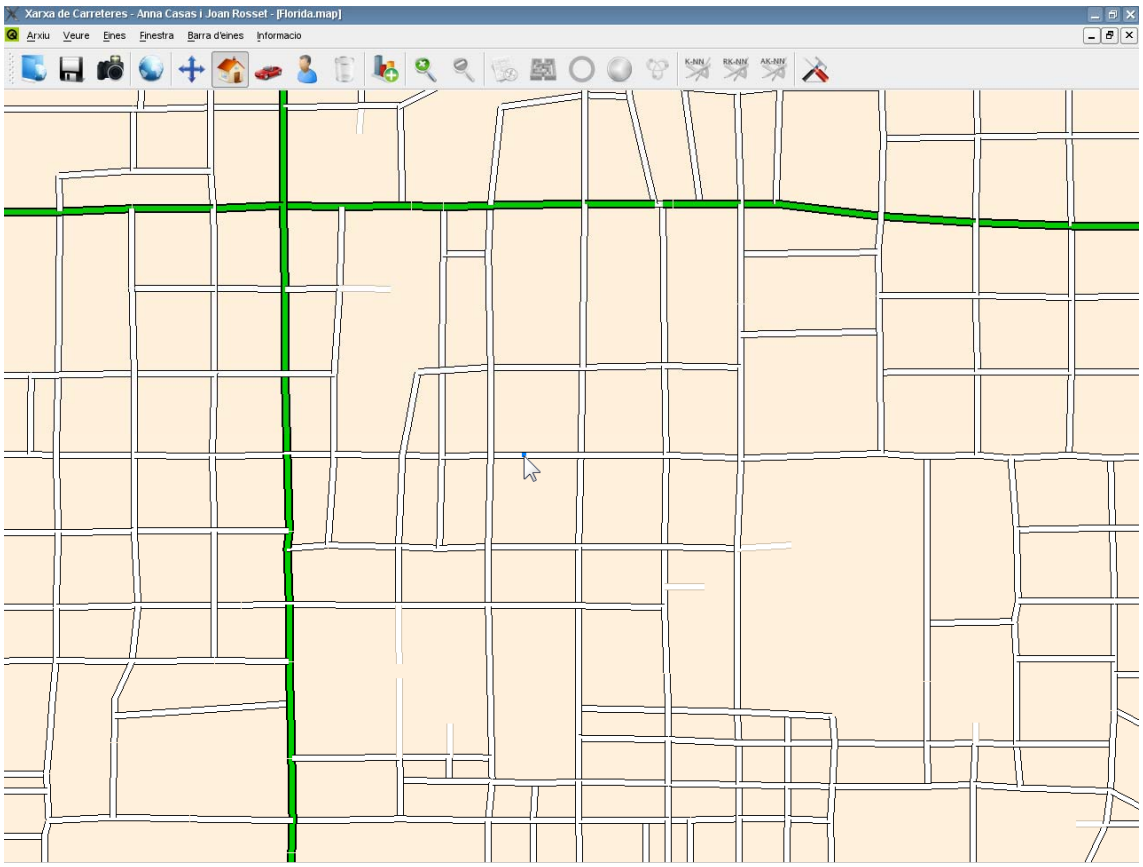


Figura 10.19: Punt virtual sobre la xarxa de carreteres

- Quan hàgim decidit on s'afegirà la facilitat, premem el botó esquerre del ratolí i ens apareixerà una finestra on introduirem les seves dades *(figura 10.20)*.

Figura 10.20: Finestra per introduir dades de la facilitat

- Un cop hàgim introduït les dades, premem el botó *Introduir* i la facilitat s'afegirà automàticament a la xarxa de carreteres *(figura 10.21)*.

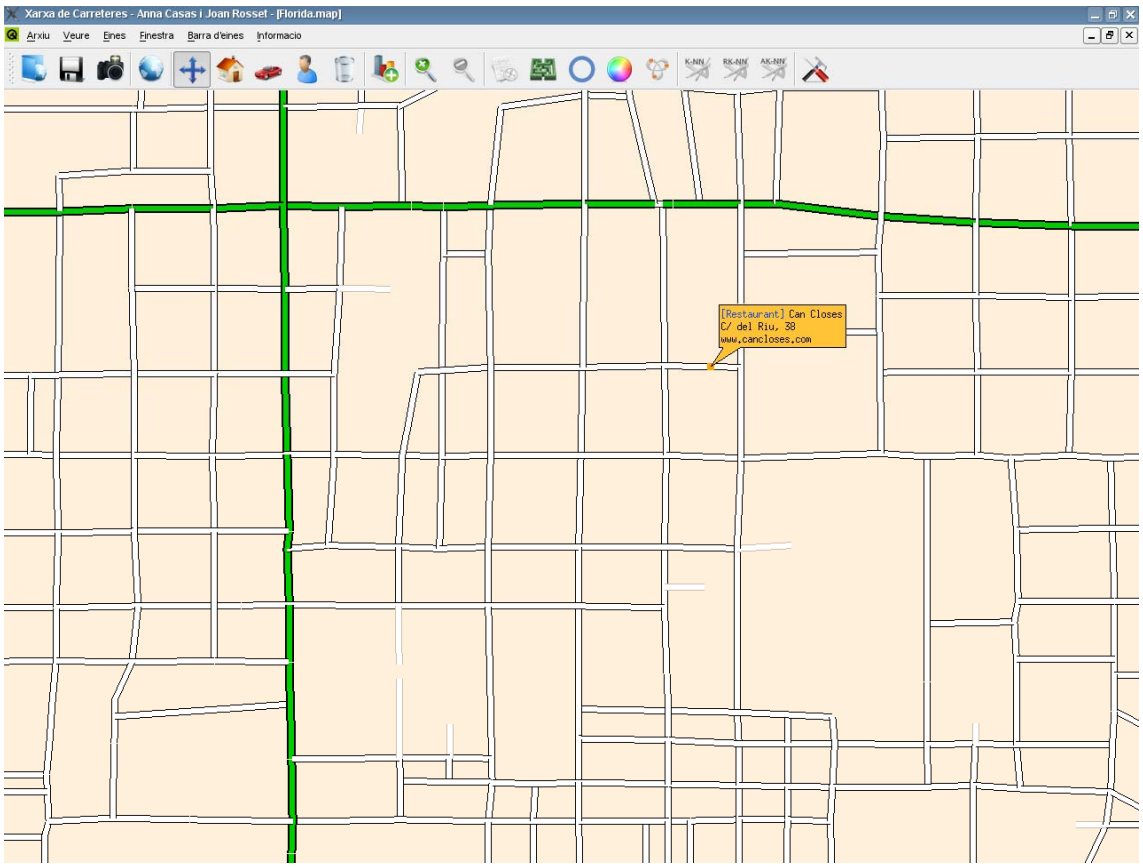


Figura 10.21: Xarxa de carreteres amb una nova facilitat

10.4.2.3. Afegir vehicle

Aquest mode permet afegir un vehicle a la xarxa de carreteres seleccionant la seva ubicació i introduint les seves dades. Aquest és molt similar a l'afegir facilitat.

A continuació es pot veure un exemple de com afegir un vehicle a una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i l'aplicació es posa automàticament en el mode d'afegir un vehicle. Ara, desplaçant el ratolí per sobre la xarxa de carreteres, ens apareix un punt virtual que ens indica a on s'afegiria si preméssim el botó esquerre del ratolí.

- Quan hàgim decidit on s'afegirà el vehicle, premem el botó esquerre del ratolí i ens apareixerà una finestra on introduirem les seves dades *(figura 10.22).*

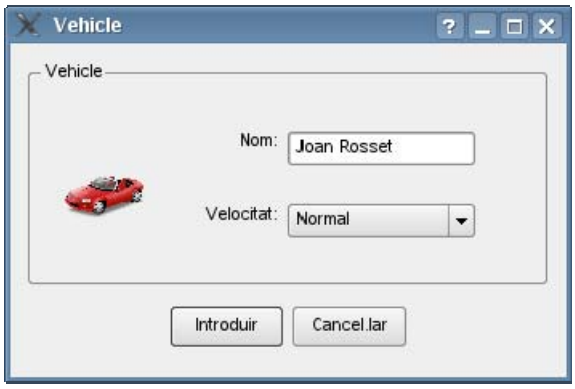


Figura 10.22: Finestra per introduir dades del car

- Un cop hàgim introduït les dades, premem el botó *Introduir* i el vehicle s'afegirà automàticament a la xarxa de carreteres. *(figura 10.23)*.

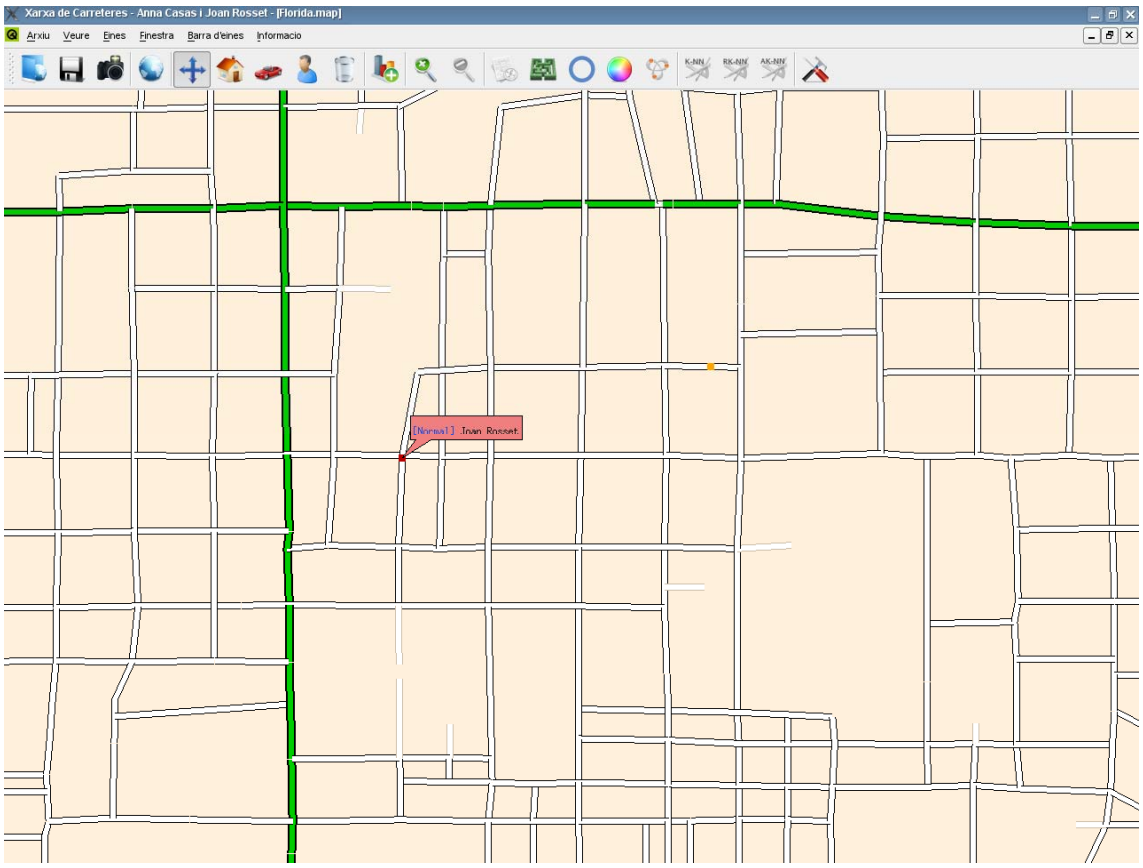


Figura 10.23: Xarxa de carreteres amb un nou vehicle

10.4.2.4. Afegir punt de consulta

Aquest mode permet afegir un punt de consulta a la xarxa de carreteres seleccionant la seva ubicació i introduint el seu nom. Aquest és molt similar a l'afegir facilitat i l'afegir vehicle.

A continuació es pot veure un exemple de com afegir un punt de consulta a una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i l'aplicació es posa automàticament en el mode d'afegir un punt de consulta. Ara, desplaçant el ratolí per sobre la xarxa de carreteres, ens apareix un punt virtual que ens indica a on s'afegiria si preméssim el botó esquerre del ratolí.

- Quan hàgim decidit on s'afegirà el punt de consulta, premem el botó esquerre del ratolí i ens apareixerà una finestra on introduirem el seu nom *(figura 10.24).*

Figura 10.24: Finestra per introduir dades del punt de consulta

- Un cop hàgim introduït el nom, premem el botó *Introduir* i el punt de consulta s'afegirà automàticament a la xarxa de carreteres. *(figura 10.25)*.

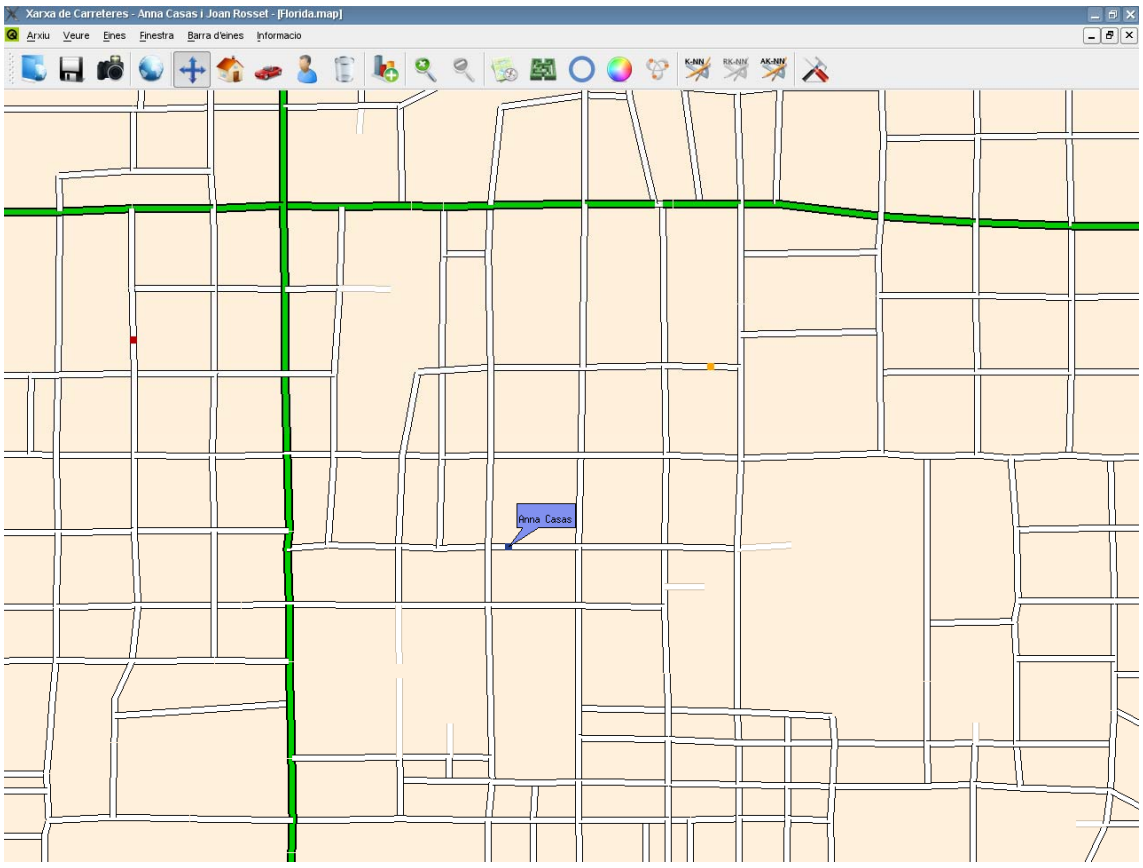


Figura 10.25: Xarxa de carreteres amb un nou punt de consulta

10.4.2.5. Eliminar objecte

Aquest mode permet eliminar qualsevol tipus d'objecte seleccionant-lo damunt la xarxa de carreteres.

A continuació es pot veure un exemple de com eliminar un objecte d'una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i l'aplicació es posa automàticament en el mode d'eliminar un objecte. Ara, desplaçant el ratolí per sobre la xarxa de carreteres, ens apareix un punt virtual que ens indica quin objecte s'eliminaria si preméssim el botó esquerre del ratolí.

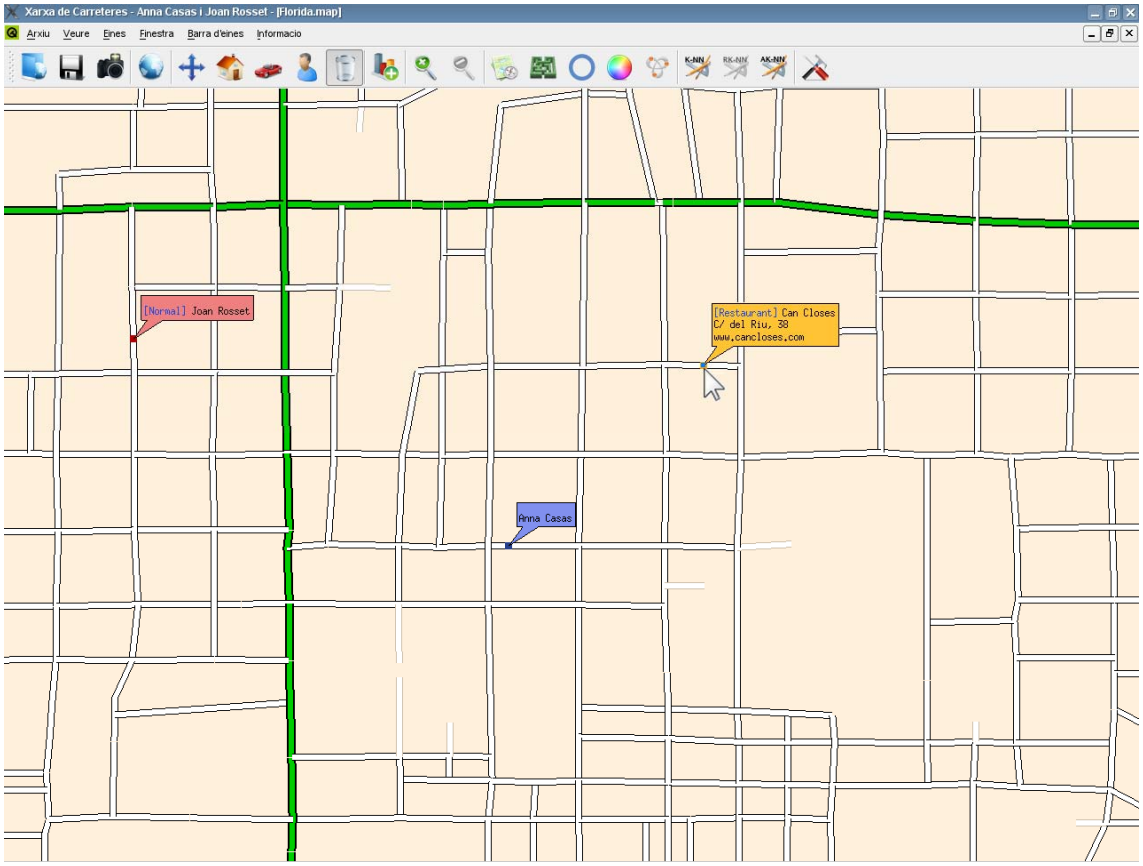


Figura 10.26: Punt virtual sobre un objecte de la xarxa de carreteres

- Quan hàgim decidit l'objecte que volem eliminar, premem el botó esquerre del ratolí i observarem com desapareix de la finestra *(figura 10.27).*

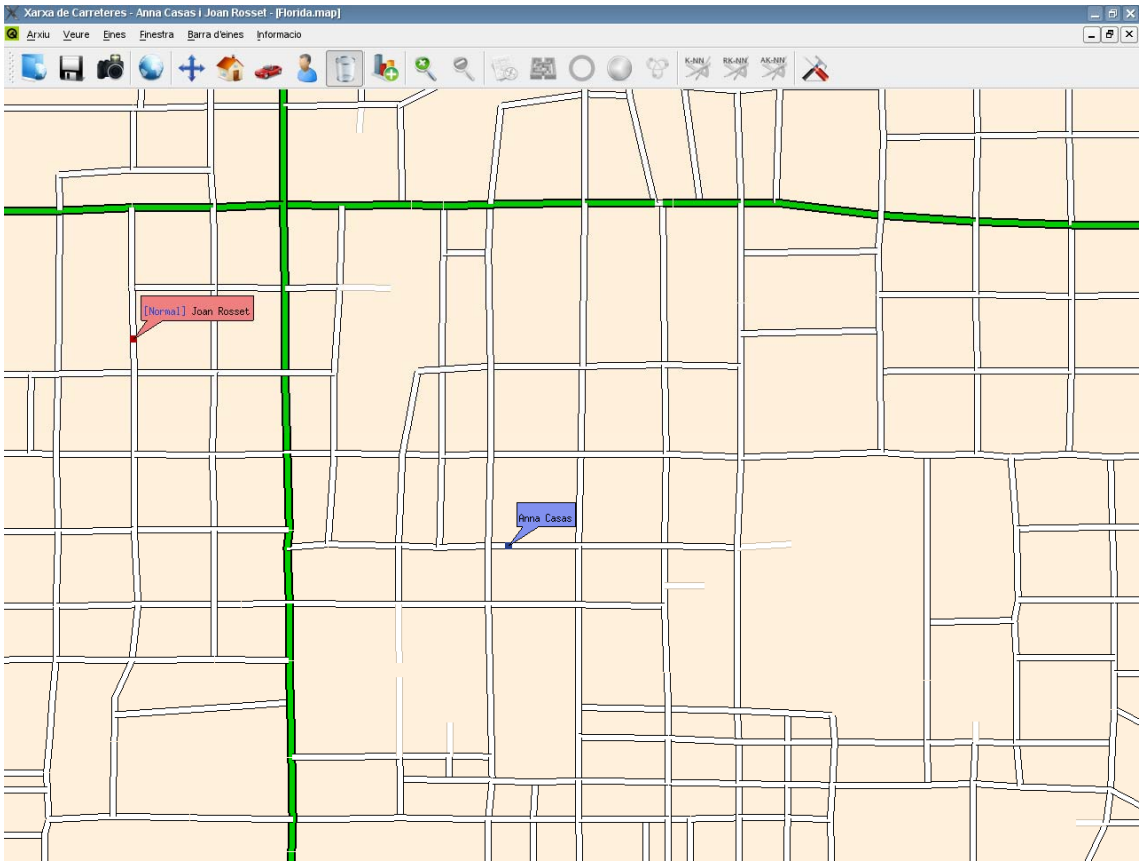


Figura 10.27: Xarxa de carreteres amb facilitat eliminada

10.4.3. Menú Eines: Afegir objectes

Aquesta opció del menú permet afegir un grup d'objectes a una xarxa de carreteres. La ubicació d'aquests nous objectes serà aleatòria.

Per a afegir-los, caldrà seleccionar quantes facilitats, vehicles i punts de consulta volem introduir. A més, en el cas de les facilitats, podrem seleccionar si volem que tinguin un tipus fix o que el seu tipus també sigui aleatori. En el cas dels vehicles també podrem escollir si volem que tinguin una velocitat fixa o que sigui aleatòria.

A continuació es pot veure un exemple de com afegir alguns objectes a una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.28)* que permet escollir quants objectes volem introduir de cada tipus. A més, també ens deixa escollir el tipus de les facilitats o la velocitat dels vehicles.

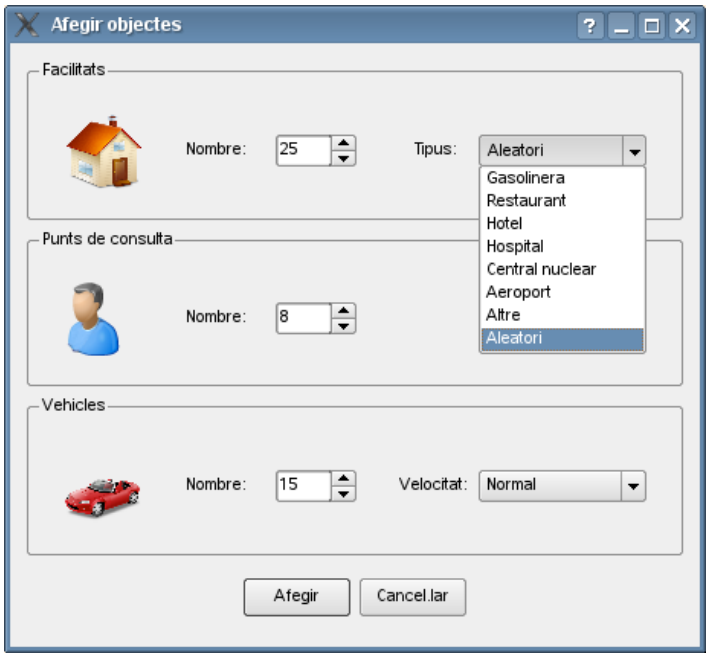


Figura 10.28: Finestra per afegir objectes

- Quan hàgim omplert les dades a la finestra anterior, premem el botó *Afegir* i automàticament podem visualitzar el resultat per pantalla *(figura 10.29)*.

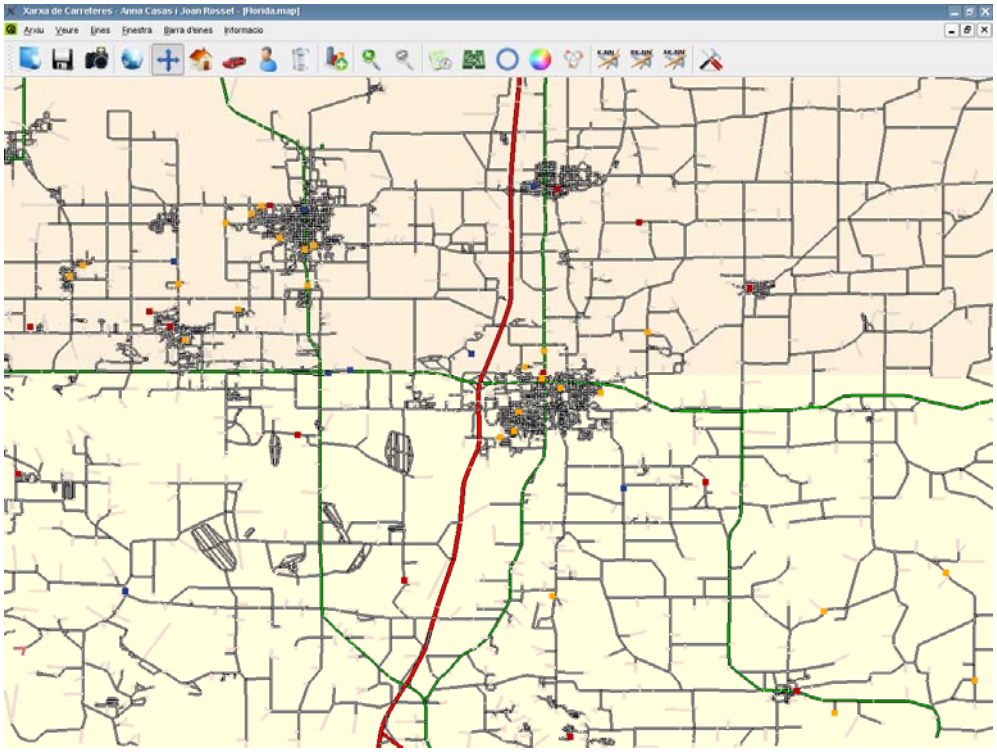


Figura 10.29: Xarxa de carreteres amb noves facilitats, punts de consulta i vehicles

10.4.4. Menú Eines: Seleccionar zona

Aquesta opció permet marcar una zona rectangular sobre la pantalla i crear una subxarxa amb les carreteres que hi hagi dins. A més, a la nova xarxa s'hi hauran de posar els objectes que conté la xarxa original.

A continuació es pot veure un exemple de com seleccionar una zona a una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i l'aplicació es posa automàticament en un mode temporal de seleccionar una zona. Ara, amb el botó esquerre del ratolí marquem un punt sobre la pantalla (una cantonada del rectangle) i, sense deixar-lo anar, ens desplacem formant un rectangle.

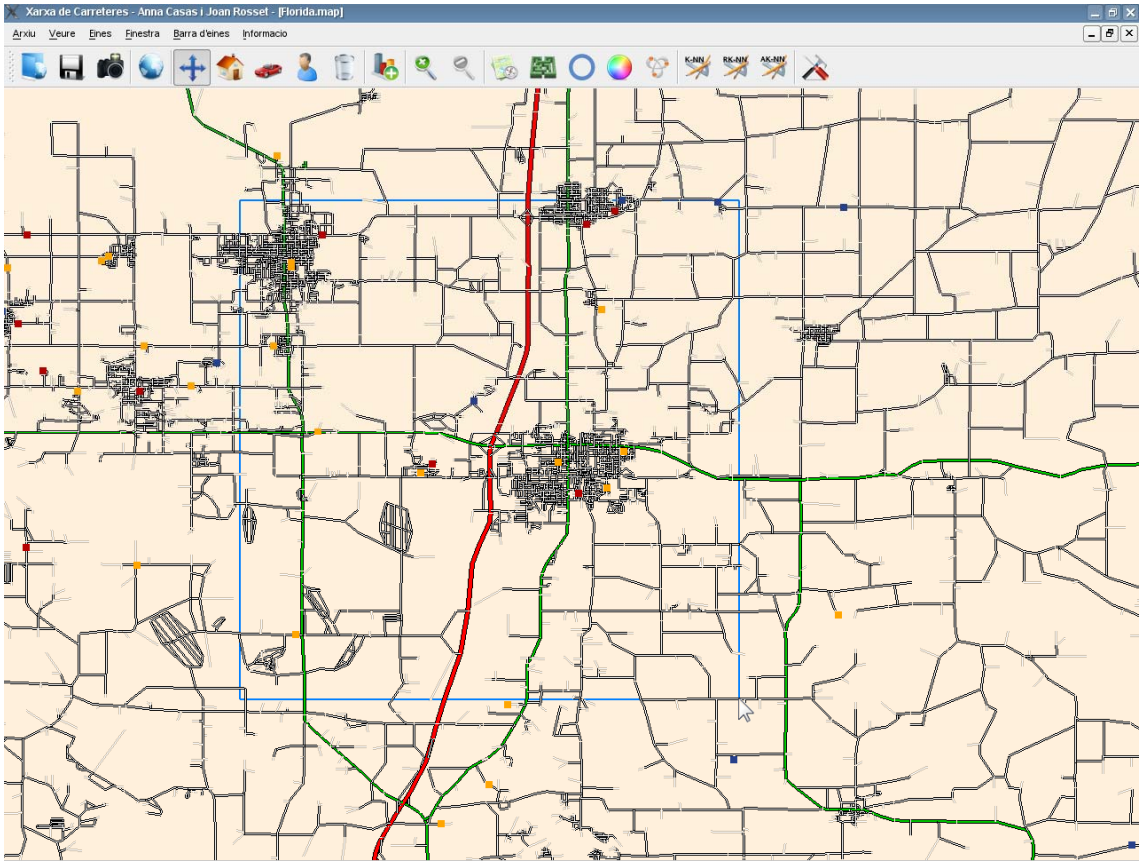


Figura 10.30: Xarxa de carreteres mentre es selecciona una zona

- Quan ja tinguem el rectangle seleccionat, deixem de prémer el botó del ratolí i automàticament l'aplicació ens mostrarà la subxarxa creada *(figura 10.31)*. Podem fixar-nos com també ha mantingut els objectes que hi havia creats.

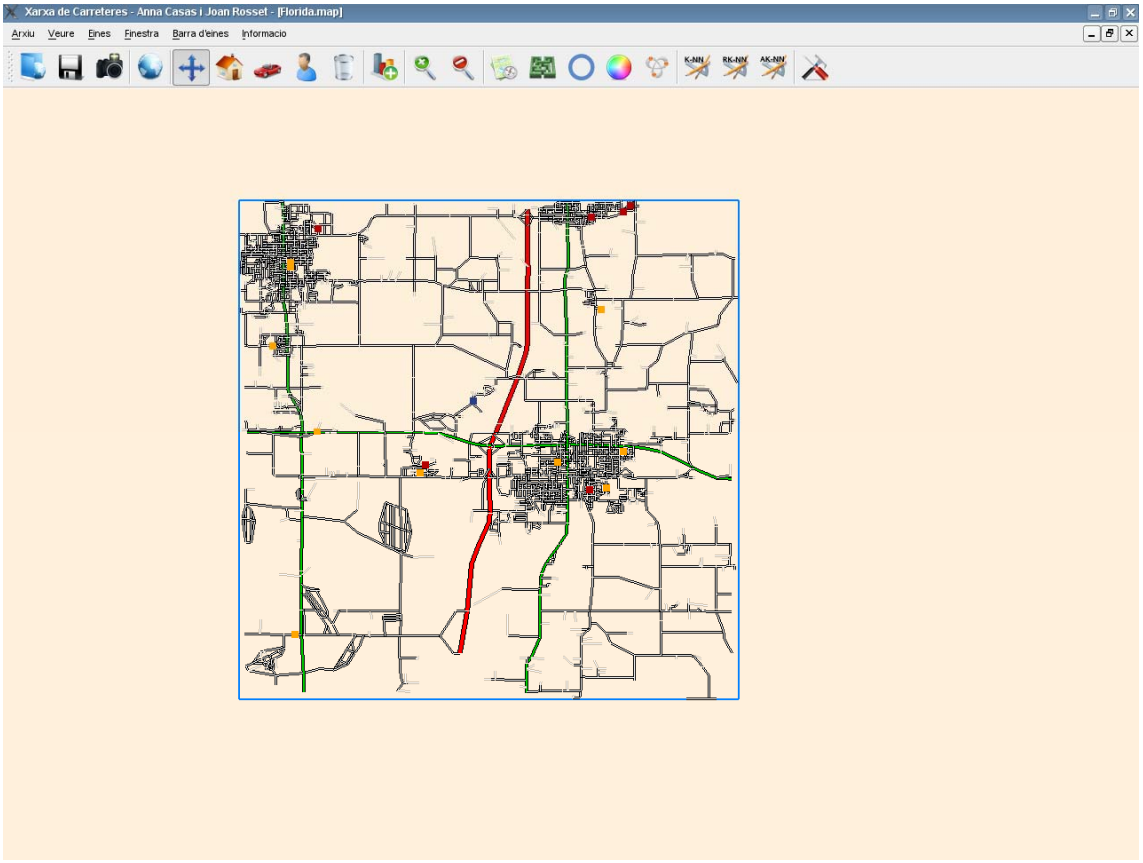


Figura 10.31: Subxarxa de carreteres

Nota: Si ens fixem amb la subxarxa que ha creat, podem veure que hi ha alguns trams de carreteres que han estat eliminats. Per exemple, el tram inferior de la autopista marcada de color vermell. Aquests trams han estat eliminats perquè per a resoldre problemes de proximitat dins la zona no es necessiten les carreteres que van a l'exterior del rectangle.

Nota: Quan tinguem una subxarxa calculada, podem tornar a utilitzar la opció *seleccionar zona* dins d'aquesta.

10.4.5. Menú Eines: Tornar al mapa original

Aquesta acció només estarà habilitada quan prèviament l'usuari hagi marcat una zona. Quan premem aquesta icona es tornarà a visualitzar la xarxa de carreteres original.

10.4.6. Menú Eines: Dijkstra

Aquesta opció permet trobar el camí mínim entre dos punts de la xarxa de carreteres. Per a fer-ho haurem de seleccionar un punt de consulta inicial i una facilitat com a punt de destí. Òbviament, l'algorisme que s'utilitza per a trobar el camí mínim és l'algorisme de Dijkstra.

A continuació es pot veure un exemple de com calcular un camí mínim en una xarxa de carreteres mitjançant aquesta opció:

- Premem la icona i ens apareix una finestra que ens informa del que cal fer per a utilitzar aquesta opció.

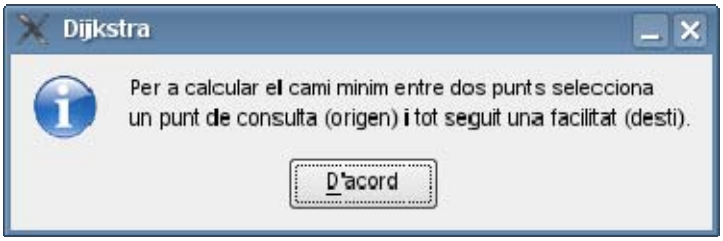


Figura 10.32:: Finestra informativa de calcular camí mínim

- Ens desplacem amb el ratolí per la xarxa de carreteres i podem observar que apareix un punt virtual sobre els punts de consulta quan passem per damunt d'ells. Aquest punt ens ajuda a seleccionar el punt que volem, ja que si n'hi ha de molt junts, podem veure quin seleccionem abans de fer-ho. Tot seguit marquem el punt de consulta inicial.

- Ara el punt virtual ens apareix quan passem amb el ratolí per damunt de cada facilitat. Igual que en el cas anterior, marquem la facilitat que desitgem. Finalment, es mostrarà sobre la xarxa de carreteres el resultat *(figura 10.33).*

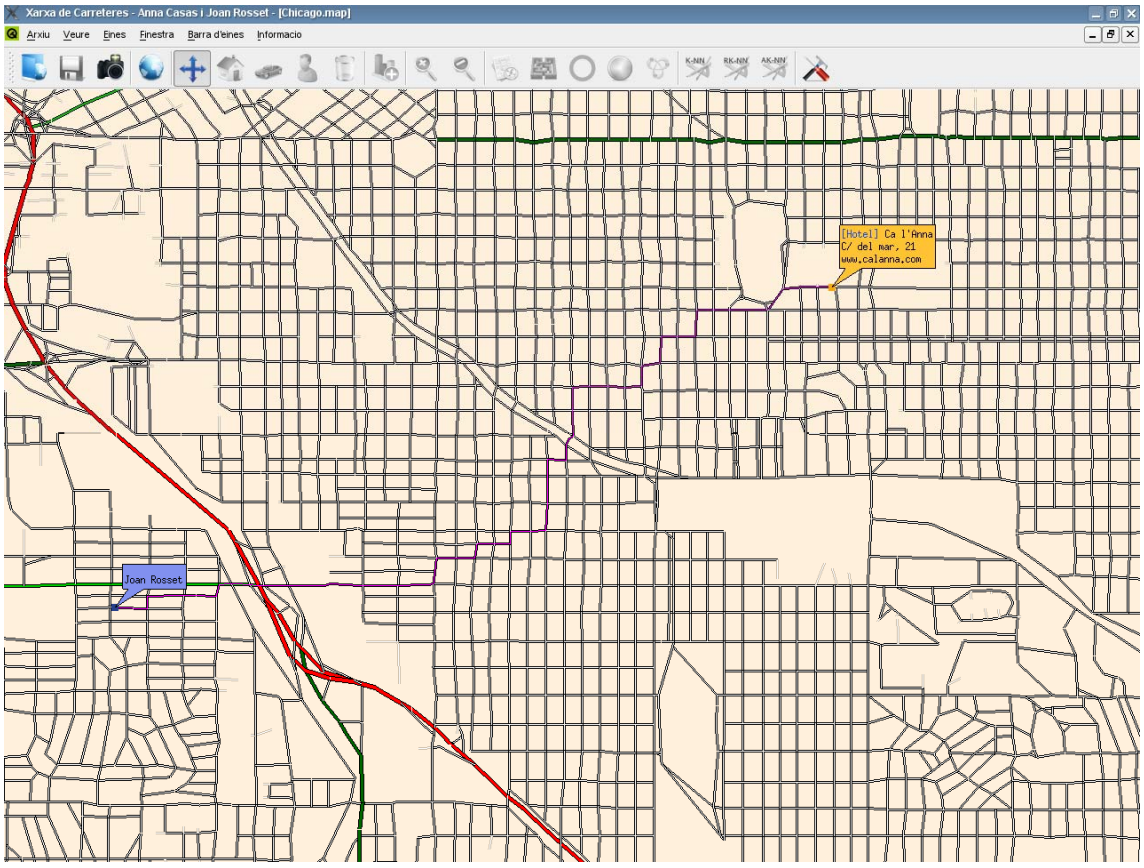


Figura 10.33: Resultat de calcular un camí mínim amb Dijkstra

Nota: Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.7. Menú Eines: Camps distàncies

Aquesta opció permet calcular la funció camps distàncies sobre una facilitat en una xarxa de carreteres.

A continuació es pot veure un exemple de com calcular la funció camps distàncies mitjançant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.34)* que ens informa del que cal fer per a utilitzar aquesta opció.

Figura 10.34: Finestra informativa de calcular la funció camps distàncies

- Ens desplacem amb el ratolí per la xarxa de carreteres i podem observar que apareix un punt virtual sobre les facilitats quan passem per damunt d'elles. Aquest punt ens ajuda a seleccionar la facilitat que volem, ja que si n'hi ha de molt juntes, podem veure quina seleccionem abans de fer-ho.

- Marquem la facilitat que desitgem i tot seguit es mostrarà sobre la xarxa de carreteres el resultat *(figura 10.35).*

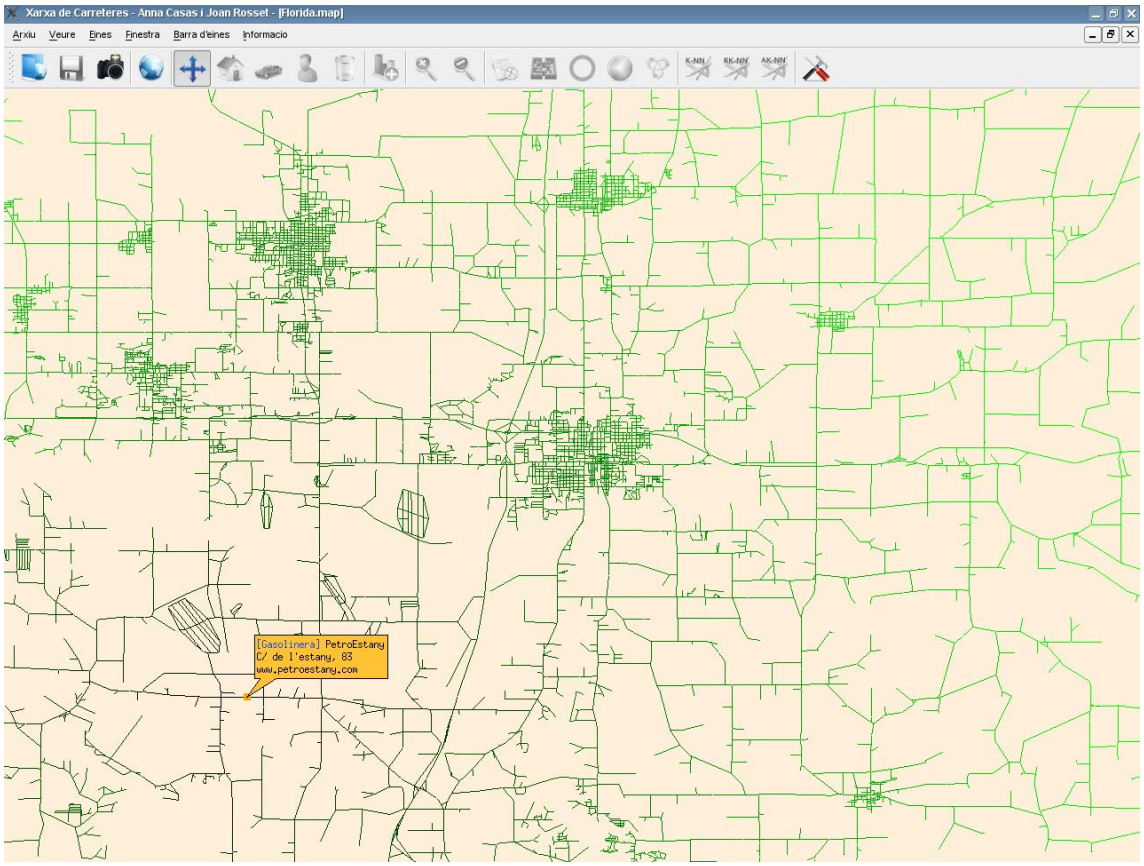


Figura 10.35: Resultat de calcular la funció camps distàncies

Nota: Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.8. Menú Eines: Marcar cercle

Aquesta opció permet calcular una zona similar a un cercle en una xarxa de carreteres. El centre d'aquesta zona serà una facilitat i el radi una distància donada. Cal remarcar que aquesta funcionalitat no calcula un cercle exacte, sinó que calcula tots els trams de carreteres que estan a una distància menor al radi de la facilitat. La forma resultant és similar a un cercle, i d'aquí el seu nom.

Aquesta opció també selecciona totes les facilitats que es troben dins de la zona delimitada pel cercle.

A continuació es pot veure un exemple de com calcular-lo en una xarxa de carreteres utilitzant aquesta opció:

- Premem la icona i ens apareix una finestra que ens informa del que cal fer per a utilitzar aquesta opció.

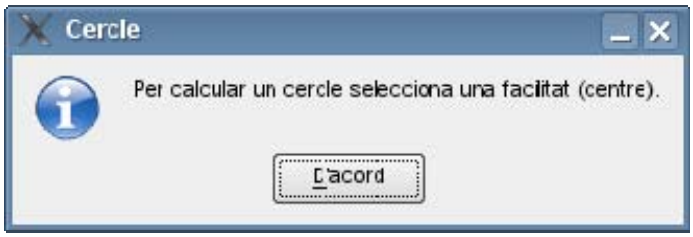


Figura 10.36: Finestra informativa de calcular un cercle

- Ens desplacem amb el ratolí per la xarxa de carreteres i podem observar que apareix un punt virtual sobre les facilitats quan passem per damunt d'elles. Aquest punt ens ajuda a seleccionar la facilitat que volem, ja que si n'hi ha de molt juntes, podem veure quina seleccionem abans de fer-ho.

- Marquem la facilitat que desitgem i tot seguit es mostrarà una finestra *(figura 10.37)* on introduirem el radi de la zona circular.

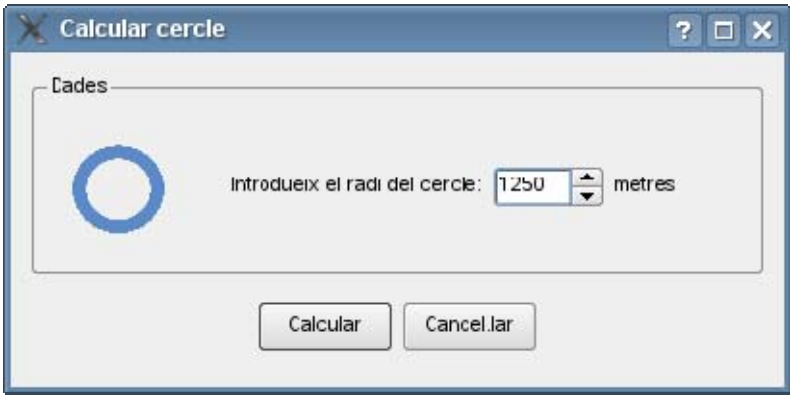


Figura 10.37: Finestra per introduir el radi de la zona circular

- En el moment que premem el botó *Calcular* ens apareix el resultat sobre la xarxa de carreteres. A la *figura 10.38* podem veure el resultat de calcular una zona circular sense facilitats i a la *figura 10.39* el resultat de calcular-la amb un grup de facilitats.

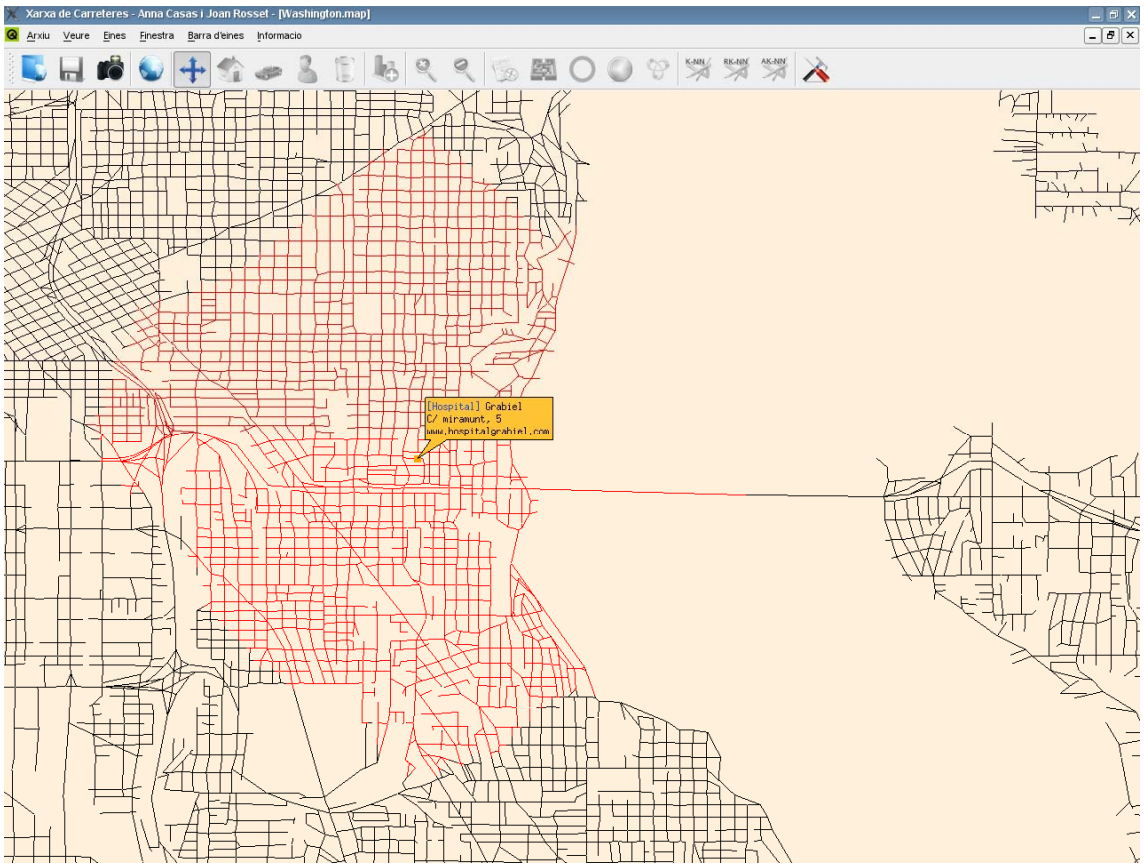


Figura 10.38: Resultat de calcular la zona circular sense facilitats

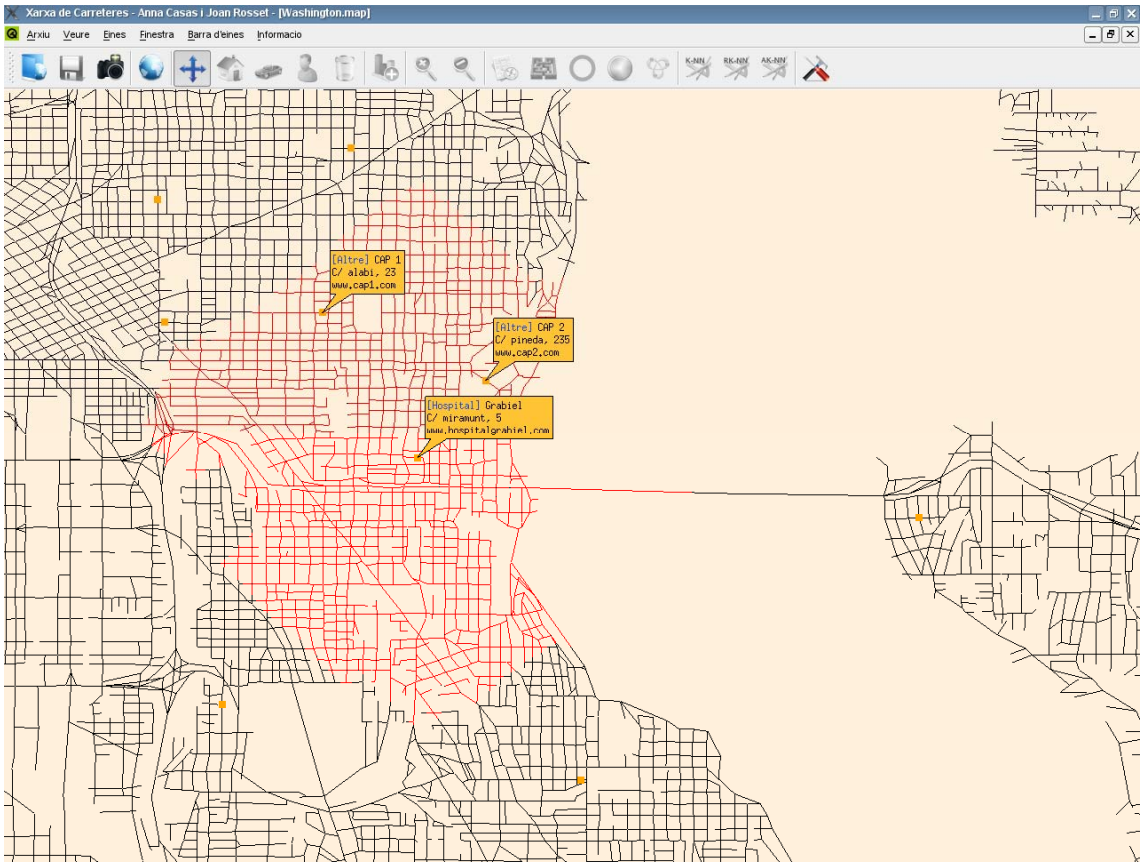


Figura 10.39: Resultat de calcular la zona circular amb facilitats

Nota: Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.9. Menú Eines: Voronoi

Aquesta opció permet calcular diagrames de Voronoi a partir de les facilitats d'una xarxa de carreteres. Es poden calcular diferents tipus de diagrames de Voronoi:

- Diagrama de Voronoi proper.
- Diagrama de Voronoi llunyà.
- Diagrama de Voronoi d'ordre k (es pot escollir visualitzar el diagrama d'ordre k o només la capa k).

A continuació es pot veure un exemple de com calcular-los en una xarxa de carreteres utilitzant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.40)* que ens permet escollir quin tipus de diagrama de Voronoi volem construir. En el cas que en vulguem construir un d'ordre k o capa k, haurem d'introduir aquest valor 'k'.

Figura 10.40: Finestra per a escollir el tipus de diagrama de Voronoi

- En el moment que premem el botó *Calcular* ens apareix el resultat sobre la xarxa de carreteres. A la *figura 10.41* podem veure el resultat de calcular un diagrama de Voronoi proper i a la *figura 10.42* el resultat de calcular el llunyà.

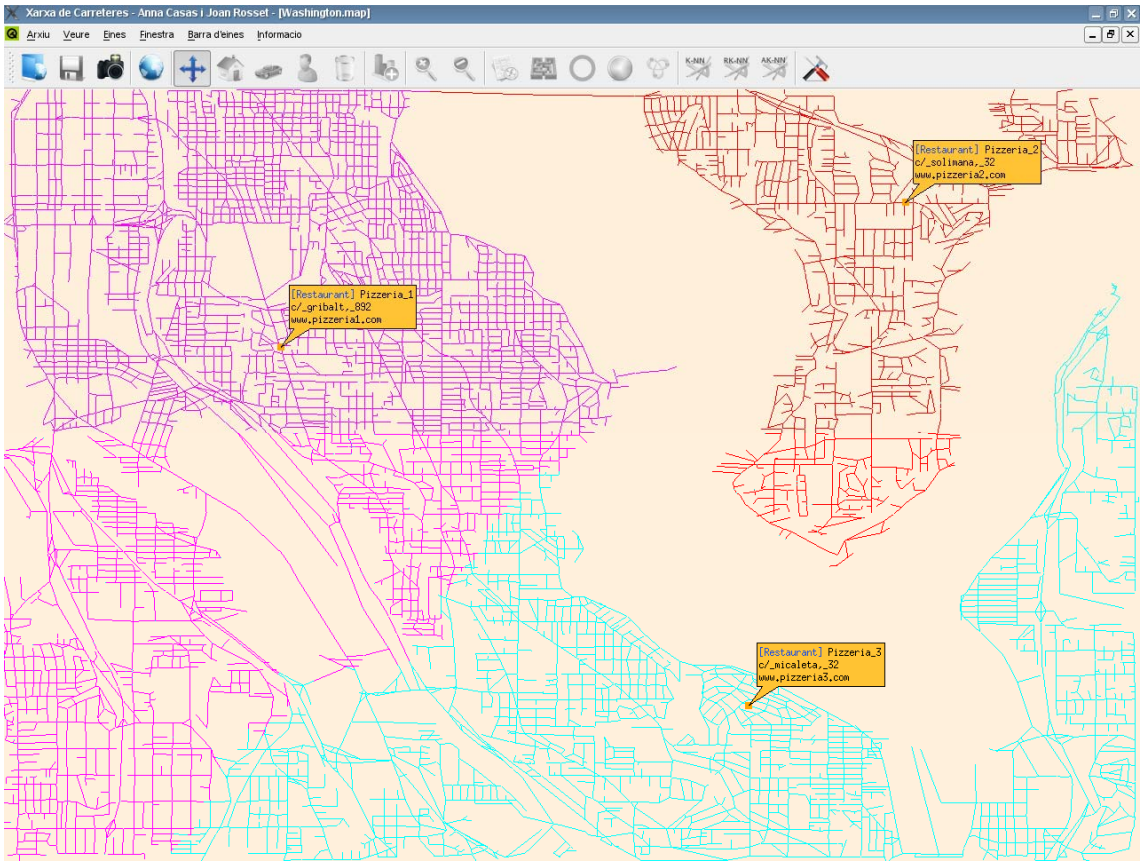


Figura 10.41: Resultat de calcular el diagrama de Voronoi proper

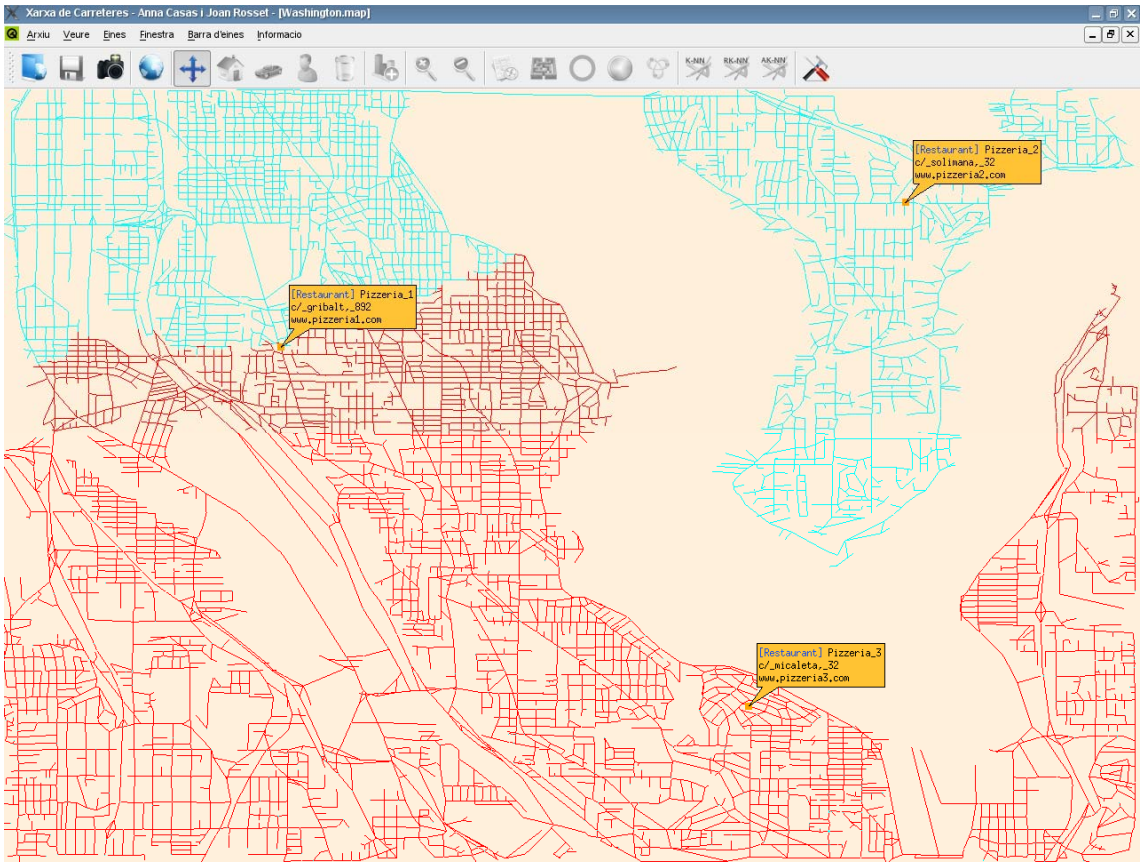


Figura 10.42: Resultat de calcular el diagrama de Voronoi llunyà

Nota: Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.10. Menú Eines: Punts d'interès

Aquesta opció permet calcular quatre punts d'interès sobre una xarxa de carreteres:

- 1-Median.
- 1-Center.
- 1-Median Obnoxious.
- 1-Center Obnoxious.

A continuació es pot veure un exemple de com calcular els punts d'interès d'una xarxa de carreteres mitjançant aquesta opció:

- Hem de prémer la icona i automàticament l'aplicació realitza els càlculs per a trobar els punts. Tot seguit ens mostra els punts trobats en una finestra *(figura 10.43)* .

Figura 10.43: Resultat de calcular els punts d'interès

- Els punts també apareixen sobre la xarxa de carreteres *(figura 10.44)*.

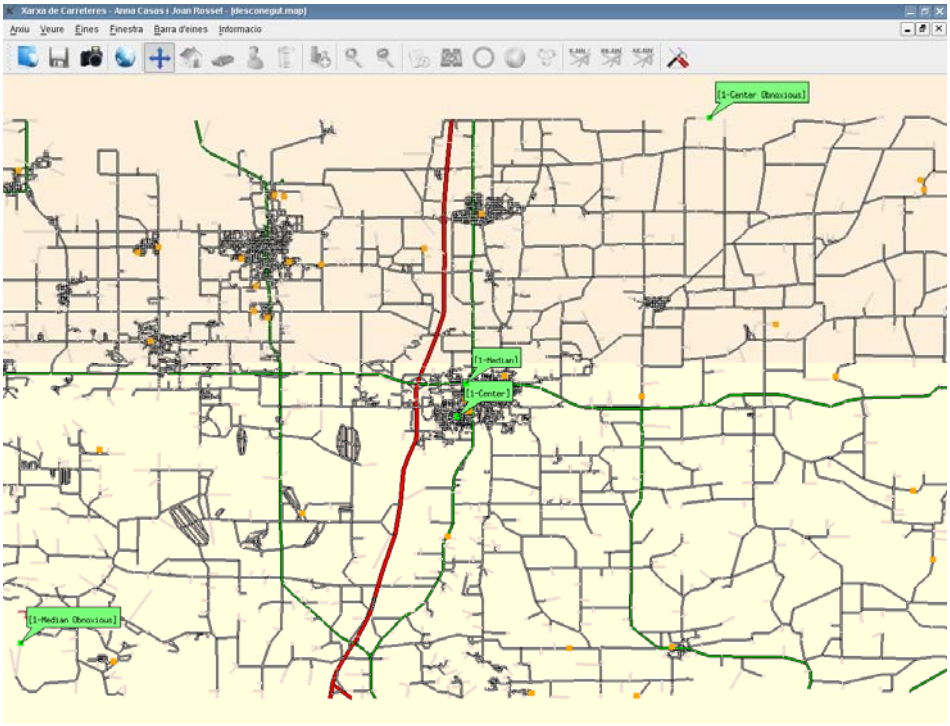


Figura 10.44: Resultat de calcular els punts d'interès

10.4.11. Menú Eines: k-NN

Aquesta opció permet plantejar el problema de proximitat k-veïns més propers i resoldre'l.

A continuació es pot veure un exemple de com resoldre aquest tipus de problema en una xarxa de carreteres utilitzant aquesta opció:

- Premem la icona i ens apareix una finestra que ens informa del que cal fer per a utilitzar aquesta opció.

Figura 10.45: Finestra informativa per plantejar problema k-NN

- Ens desplacem amb el ratolí per la xarxa de carreteres i podem observar que apareix un punt virtual sobre els punts de consulta quan passem per damunt d'ells. Aquest punt virtual ens ajuda a seleccionar el punt de consulta que volem, ja que si n'hi ha de molt junts, podem veure quin seleccionem abans de fer-ho.

- Marquem el punt de consulta que desitgem i tot seguit es mostrarà una finestra *(figura 10.46)* on plantejarem el problema.

Figura 10.46: Finestra per a plantejar el problema k-NN

- En el moment que premem el botó *Buscar* l'aplicació resol el problema i mostra les facilitats resultants en una llista dins una finestra *(figura 10.47).*

Figura 10.47: Resultat de resoldre un problema k-NN

- Les facilitats resultants també apareixen sobre la xarxa de carreteres *(figura 10.48)*.

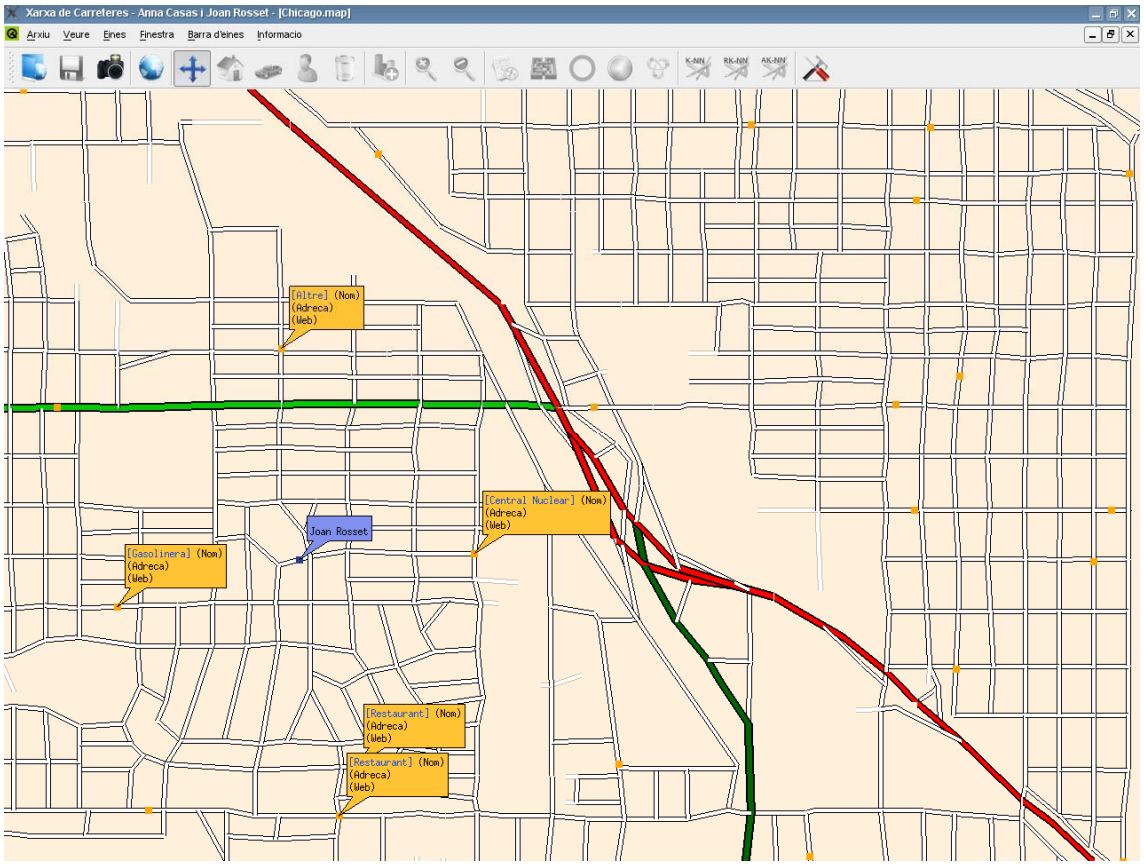


Figura 10.48: Resultat de resoldre un problema k-NN

Nota: Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.12. Menú Eines: Rk-NN

Aquesta opció permet plantejar el problema de proximitat k-veïns més propers inversos i resoldre'l.

A continuació es pot veure un exemple de com resoldre aquest tipus de problema en una xarxa de carreteres utilitzant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.49)* que ens informa del que cal fer per a utilitzar aquesta opció.

Figura 10.49: Finestra informativa per plantejar problema Rk-NN

- Ens desplacem amb el ratolí per la xarxa de carreteres i podem observar que apareix un punt virtual sobre les facilitats quan passem per damunt d'elles. Aquest punt virtual ens ajuda a seleccionar la facilitat que volem, ja que si n'hi ha de molt juntes, podem veure quina seleccionem abans de fer-ho.

- Marquem la facilitat que desitgem i tot seguit es mostrarà una finestra *(figura 10.50)* on plantejarem el problema.

Figura 10.50: Finestra per a plantejar el problema Rk-NN

- En el moment que premem el botó *Buscar* l'aplicació resol el problema i mostra les facilitats resultants en una llista dins una finestra *(figura 10.51)*.

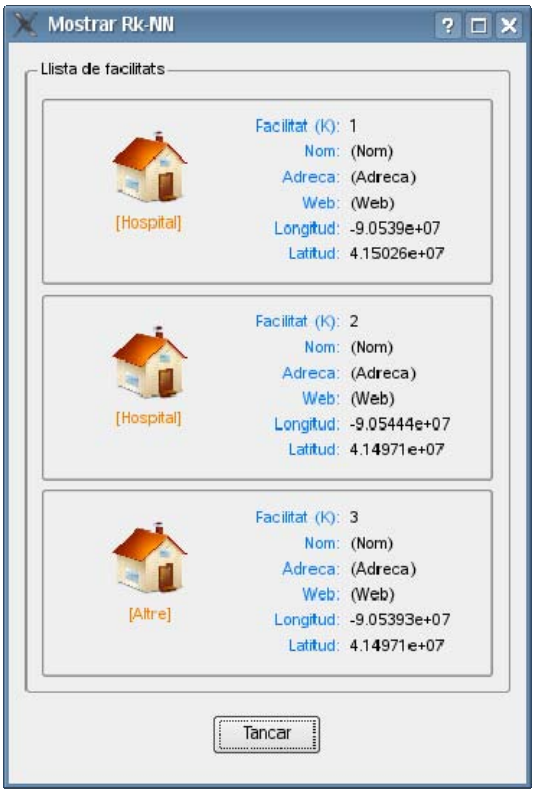


Figura 10.51: Resultat de resoldre el problema Rk-NN

- Les facilitats resultants també apareixen sobre la xarxa de carreteres *(figura 10.52)*. Per a ajudar la comprensió, s'ha senyalat la facilitat que hem marcat inicialment per a resoldre el problema.

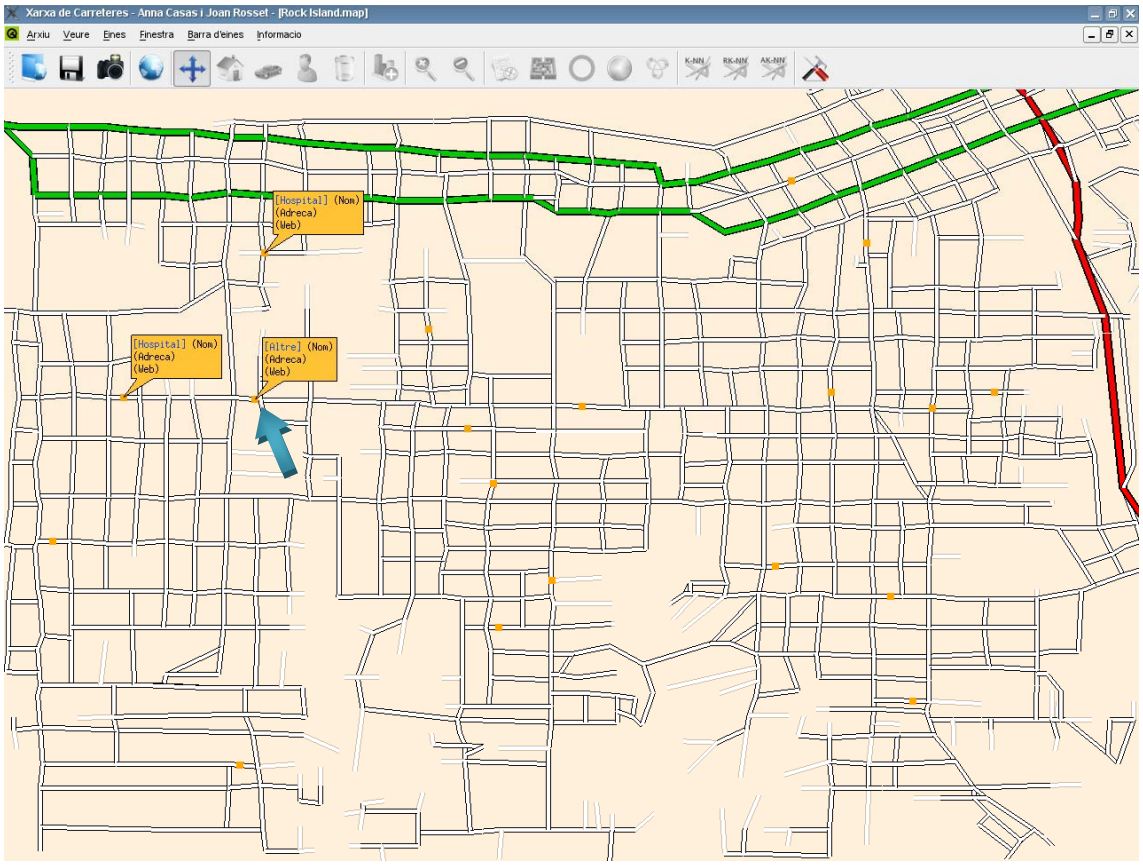


Figura 10.52: Resultat de resoldre el problema Rk-NN

Nota: Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.13. Menú Eines: Ak-NN

Aquesta opció permet plantejar el problema de proximitat k-veïns més propers agregats i resoldre'l. Existeixen diferents tipus de problemes:

- MinMax
- MinSum
- MaxMin
- MaxSum

A continuació es pot veure un exemple de com resoldre aquest tipus de problema en una xarxa de carreteres utilitzant aquesta opció:

- Premem la icona i ens apareix una finestra *(figura 10.53)* on plantejarem el problema.

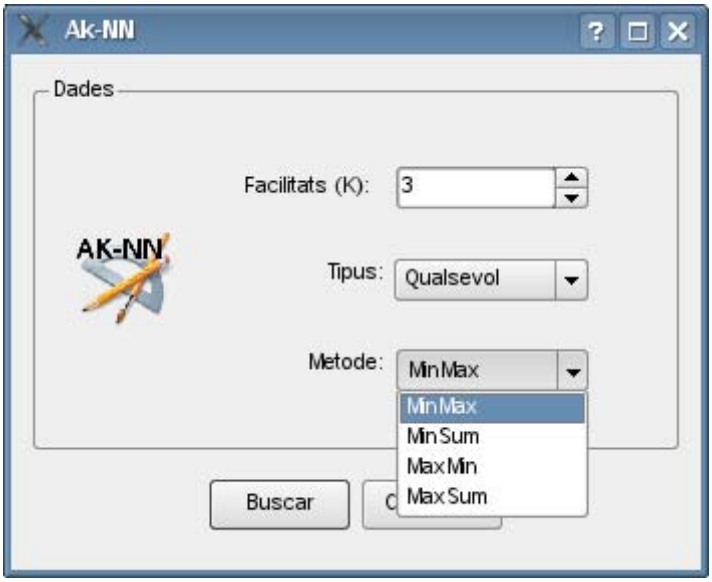


Figura 10.53: Finestra per a plantejar el problema Ak-NN

- En el moment que premem el botó *Buscar* l'aplicació resol el problema i mostra les facilitats resultants en una llista dins una finestra *(figura 10.54).*

Figura 10.54: Resultat de resoldre un problema Ak-NN

- Les facilitats resultants també apareixen sobre la xarxa de carreteres *(figura 10.55)*.

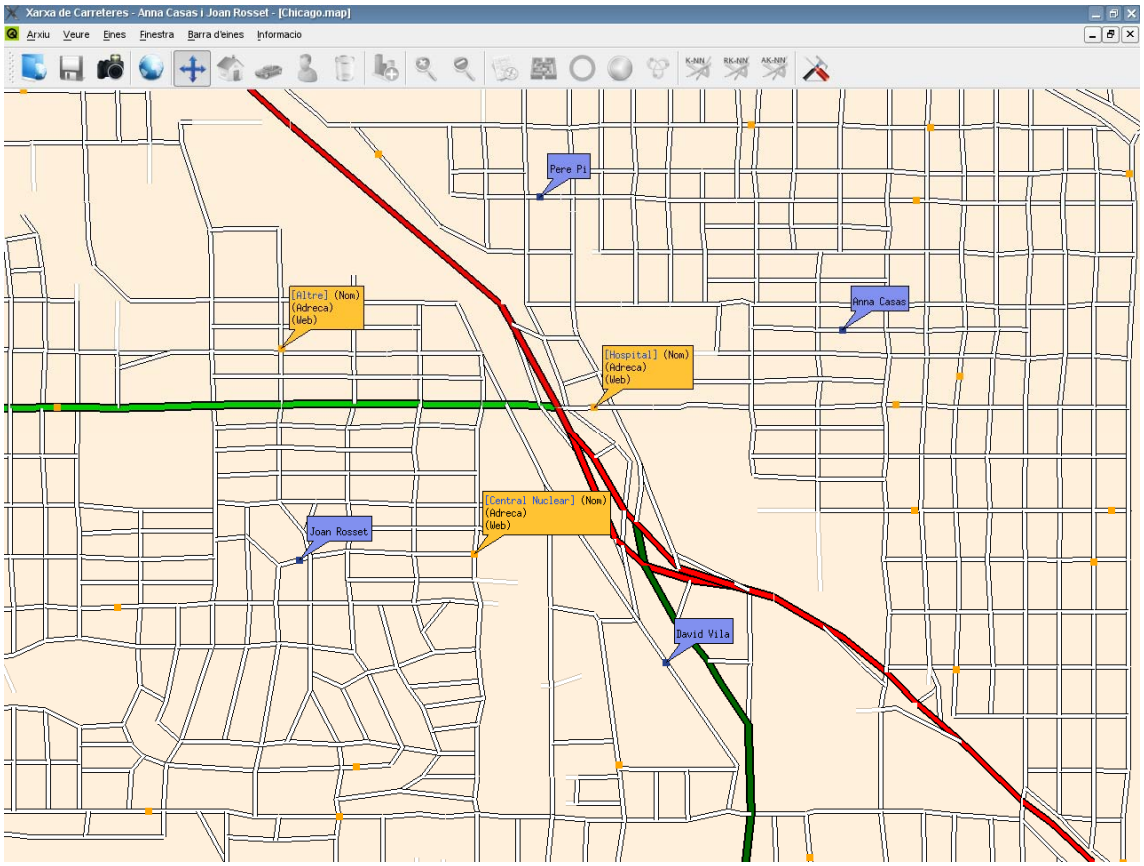


Figura 10.55: Resultat de resoldre un problema Ak-NN

- **Nota:** Mentre visualitzem el resultat, podem observar com gairebé totes les opcions de l'aplicació queden inhabilitades. D'aquesta manera s'evita barrejar problemes. Per a calcular un nou problema s'ha d'anar a la opció *veure mapa*, que reinicialitza la xarxa de carreteres.

10.4.14. Menú Eines: Preferències

Aquesta opció permet canviar les preferències de l'aplicació. Tenim dos preferències:

- Veure el mapa detalladament o en un mode simplificat.
- Ajudar a seleccionar objectes o seleccionar-los sense ajuda.

A continuació es pot veure un exemple d'un canvi de preferències:

- Premem la icona i ens apareix una finestra *(figura 10.56)* que permet canviar les preferències de l'aplicació.

Figura 10.56: Finestra per a canviar les preferències

- Un cop hàgim canviat les preferències, premem el botó *Desar* i s'aplicaran automàticament. Tot seguit podem veure una xarxa de carreteres amb el mapa detallat i sense el mapa detallat.

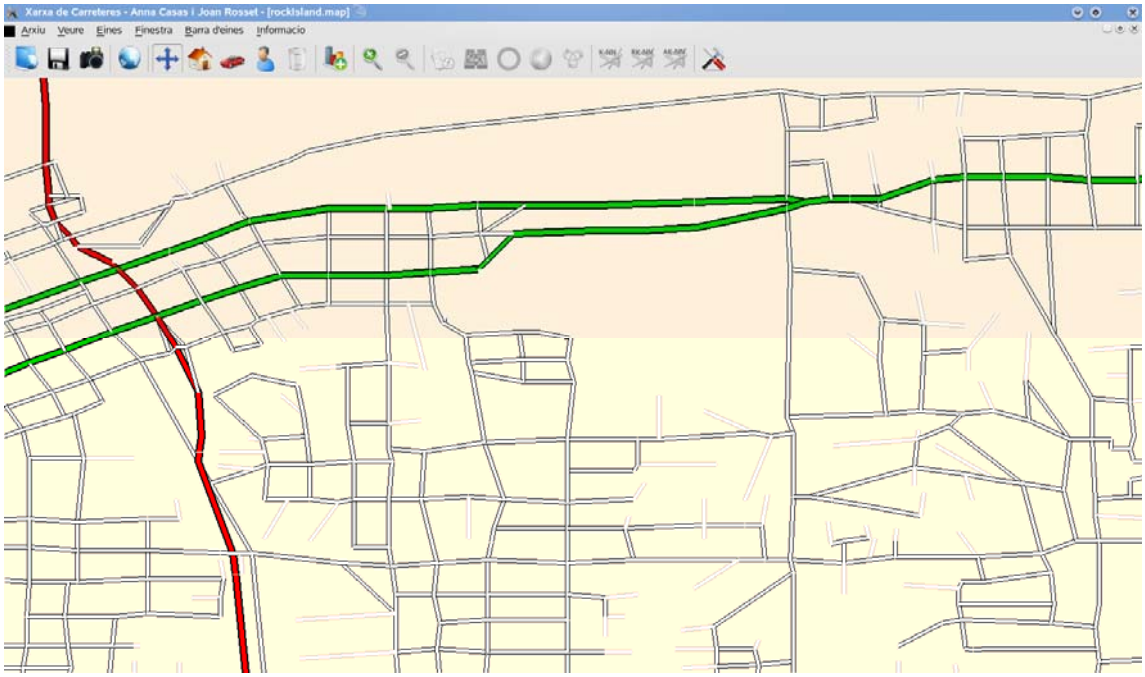


Figura 10.57: Xarxa de carreteres detallada

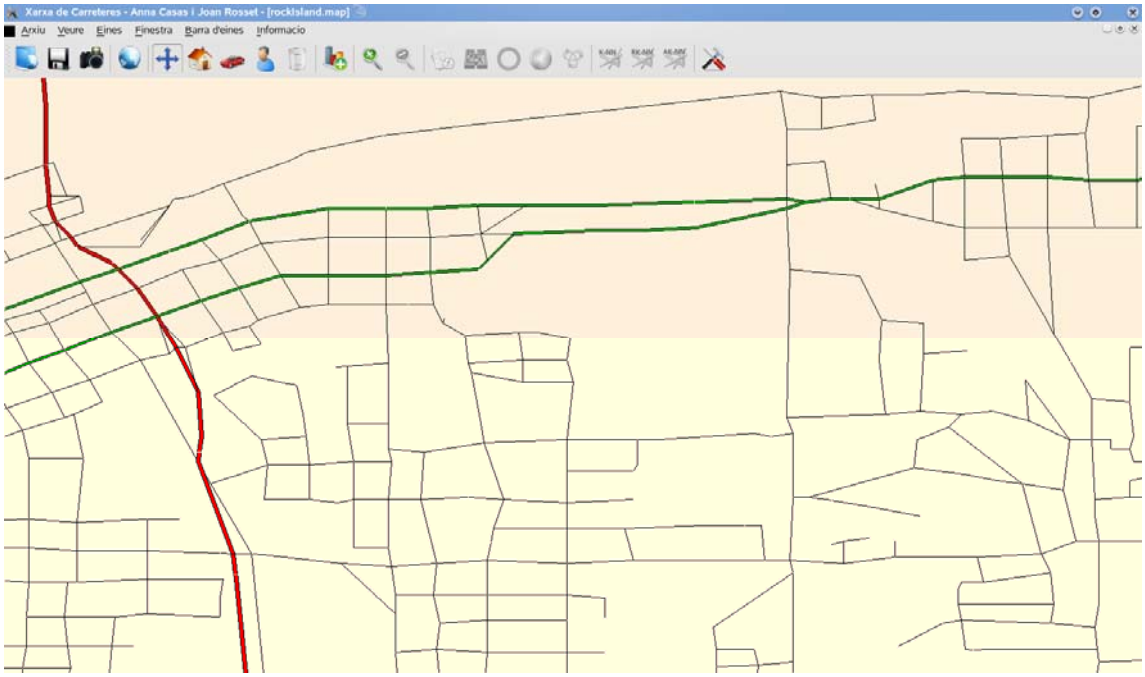


Figura 10.58: Xarxa de carreteres no detallada

10.5. Menú: Finestra

El menú Finestra, a diferència dels altres menús, no conté ítems on cadascun d'ells realitza una funció determinada. Aquest menú conté tants ítems com finestres té la nostra aplicació i a través d'aquests es pot activar la finestra desitjada. Si observem la *figura* podem veure que l'aplicació té dos finestres obertes:

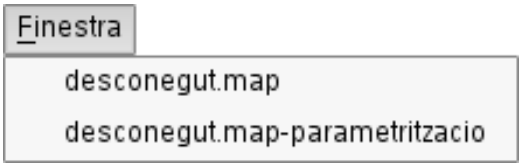


Figura 10.59: Menú: Finestra

10.6. Menú: Barra d'eines

El menú Barra d'eines conté totes aquelles accions que es poden realitzar sobre la barra d'eines de l'aplicació. Tal i com es mostra a la *figura*, té els següents ítems:

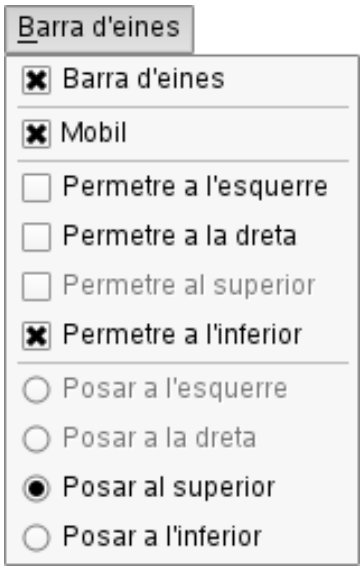


Figura 10.60: Menú: Barra d'eines

- **Barra d'eines:** Ítem per a mostrar o ocultar la barra d'eines.
- **Mòbil:** Ítem que permet a l'usuari moure la barra d'eines o bé deixar-la permanentment en una ubicació determinada.
- **Permetre a esquerra, dreta, superior i inferior:** Aquests quatre ítems permeten a l'usuari indicar a quins dels quatre cantons de l'aplicació pot estar la barra d'eines. Cal remarcar que es pot escollir més d'una opció alhora.
- **Posar a esquerra, dreta, superior o inferior:** Aquests quatre ítems permeten a l'usuari indicar a quina posició vol ubicar la barra d'eines. Com és lògic, només podrà escollir una opció.

10.7. Menú: Informació

El menú Ajuda, tal i com es mostra a la *figura*, té el següent ítem:

Figura 10.61: Menú: Informació

10.7.1. Menú Informació: Sobre Xarxa de Carreteres

Aquesta opció el que fa és obrir una finestra mostrant a informació d'interès sobre l'aplicació *(figura 10.62)*.

Figura 10.62: Informació sobre l'aplicació Xarxa de Carreteres

11. Agraïments

Finalment, ens agradaria donar les gràcies a tota aquella gent que ens ha estat donant ànims en tot moment i ens han ajudat a aconseguir amb èxit la realització d'aquest projecte. Voldriem destacar:

- Al nostre tutor del projecte final de carrera Joan Antoni Sellarès, per haver-nos donat la oportunitat de fer aquest projecte, per guiar-nos en tot moment, pel material proporcionat i per contestar-nos a totes les nostres preguntes de forma agradable.

- A la Marta Fort per haver tingut tanta paciència amb nosaltres i haver-nos dedicat tantes hores del seu temps per a ajudar-nos.

- A en Yago Díez per haver ajudat a transformar els fitxers de xarxes.

- A en Gustavo Patow per l'interès mostrat i l'ajuda en la memòria.

- Als companys de l'aula per haver-nos donat moments de distracció.

12. Bibliografia

Per a la realització d'aquest projecte no s'ha utilitzat cap llibre en especial, ja que tota la informació que necessària s'ha trobat a la web o consultant-la directament al tutor.

Les webs visitades amb més freqüència han estat:

http://www.trolltech.com
Documentació de les Qt.

http://www.nvidia.es
Per a la cerca de documentació de les llibreries Cg.

http://www.google.com
Per a la cerca de tota mena d'informació al llarg del desenvolupament de l'aplicació.