Satakunnan ammattikorkeakoulu

OPINNÄYTETYÖ

Matti Viljanen

Matti Viljanen

# SÄHKÖINEN INFOTAULU ASIAKKAAN OPASTAMISEEN

Tietotekniikan koulutusohjelma

Ohjelmistotekniikan suuntautumisvaihtoehto

2011

samk

Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SÄHKÖINEN INFOTAULU ASIAKKAAN OPASTAMISEEN

Viljanen, Matti
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Tammikuu 2011
Kivi, Karri
Sivumäärä: 29

Asiasanat: Qt, C++, ohjelmisto, tietojärjestelmät, opastaminen, reititys

---

Tämän opinnäytetyön aiheena oli suunnitella ja toteuttaa ohjelmisto sähköiseen asiakkaan opastusjärjestelmään ohjelmiston tilaajan, Citidigi Oy, vaatimusten mukaan. Projektin ensisijainen käyttötarkoitus oli tuottaa ohjelmisto laitteeseen, joka ohjaisi asiakkaat oikeaan paikkaan rakennuksessa tai alueella. Opastamisessa hyödynnettäisiin ennalta järjestelmään lisättyjä rakennuksen tai alueen pohjapiirrosta sekä listaa alueella sijaitsevista kohteista. Käyttäjää opastettaisiin graafisesti näyttämällä pohjapiirros sekä polku kohteeseen.

Tämä opinnäytetyö keskittyy kartta- ja kohdedatan käsittelyyn ohjelman sisällä sekä sen esittämiseen ruudulla. Ohjelmisto kehitettiin Nokian Qt-ohjelmiston kehitys- ja suoritusympäristöä sekä Oraclen MySQL-tietokantaa käyttäen. Ohjelmointikielinä toimivat C++ sekä SQL.

ELECTRIC INFORMATION PANEL FOR CUSTOMER ASSISTANCE

Viljanen, Matti
Satakunta University of Applied Sciences
Information Technology
January 2011
Kivi, Karri
Pages: 29

Keywords: Qt, C++, software, datasystems, assistance, routing

---

The purpose of this thesis was to design and implement a software for electrical customer guidance system according to the specifications of the client, Citidigi Oy. The primary goal of the project was to produce the software to the device that would be placed on suitable location in the building or area. The device would guide the customers to desired location in the market using the floor plan and list about the targets of interest in the area inserted into the system beforehand. The user would receive guidance graphically by being presenting the floor plan and the path to the target.

This thesis focuses on the tasks of processing and displaying the map and target data on the screen. The software was developed using Qt framework from Nokia and MySQL database software from Oracle. The programming languages used were C++ and SQL.

SISÄLLYS

# TERMILUETTELO

| | |
|---|---|
| Algoritmi | Sarja toimintoja, jotka tähtäävät ongelman ratkaisemiseen tai yksinkertaistamiseen. |
| Funktio | Ohjelman palanen, joka ottaa syötteen, käsittelee sen ja palauttaa tuloksen. Funktiolle on tyypillistä se, että sitä käytetään useasti, mutta eri syötteellä. |
| Graafi | Abstrakti tietorakenne, joka koostuu tiedon palasista (solmu) ja niiden välisistä yhteyksistä (kaari). Muodostaa usein verkkomaisen rakenteen. |
| Hash Table | Kts. tiivistetaulu. |
| Kaari | Graafin osa, joka ilmaisee kahden tiedon palasen (solmun) keskinäistä yhteyttä. |
| Polygoni | Vapaamuotoinen monikulmio, esimerkiksi kolmio tai neliö. |
| Signaali | Ohjelmoinnissa käytettävä termi ”ärsykkeelle”, joka lähetetään tietyn tapahtuman yhteydessä, esimerkiksi hiiren klikkaus tai painikkeen painallus. Signaali lähetään aina, vaikka sitä ei olisi kytketty mihinkään toimintoon. |
| Signals and slots | Qt-ohjelmoinnissa käytetty nimi tavalle liittää lähetetty tieto tai käsky toiminnan suorittavaan funktioon. |

| | |
|---|---|
| Slotti | Ohjelmoinnissa käytettävä termi funktiolle, joka suoritetaan signaalin, ”ärsykkeen” lähettämisen jälkeen. Slottia voidaan käyttää tavallisena funktiona. Slottiin voi olla kytkettynä nolla, yksi tai useampia signaaleja. |
| Solmu | Graafin osa, joka kuvaa itsenäistä tiedon palasta, joista ongelma koostuu. |
| Säie | Prosessi, tai sen osa, jota prosessori suorittaa. Ohjelmiston toteutuksesta ja laitteistosta riippuen yhtä tai useampaa säiettä voidaan suorittaa samanaikaisesti. |
| Tiivistetaulu | Tiedontallennusrakenne, joka perustuu avaimeen ja tunnisteeseen. Tallennusavaimesta lasketaan tietyllä tavalla tarkistesumma, joka määrää itse tiedon sijainnin taulukossa. |

# 1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on käsitellä karttadatan käsittelyä, reitinlaskentaa sekä kartan graafista esittämistä ruudulla. Opinnäytetyönä toteutettu toiminnallisuus tehtiin osana ohjelmistoprojektia, jonka tavoitteena oli tuottaa ohjelmisto asiakkaan tavoitteen mukaan. Ohjelmiston päätavoite tämän opinnäytetyön osalta oli käsitellä ja näyttää loppukäyttäjälle pohjapiirros esimerkiksi myymälästä hyllyineen ja seinineen, sekä laskea ja näyttää lyhin reitti kahden pisteen välillä.

Ohjelmistoprojektin puitteissa toteutettiin monia muita vaadittavia komponentteja, mutta reitinlaskenta sekä karttadatan piirtäminen jätettiin alkuperäisen suunnitelman mukaisesti projektin loppuvaiheeseen. Syy tähän on se, että kartan käsittelyssä tultaisiin käyttämään lukuisia muita itse toteutettavia komponentteja. Kartta- ja reittikomponenteista muodostui lopulta työn monimutkaisimmat komponentit. Tässä opinnäytetyössä tarkastellaan ensisijaisesti toteutettujen kartta- ja reittikomponenttien toimintaa sekä grafiikan piirtämisen yhteydessä tarvittavaa suorituksen aikaista käsittelyä.

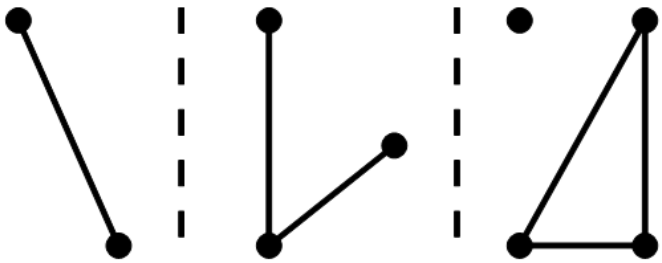
Opinnäytetyön tavoitteet olivat seuraavat:

- Tuottaa komponentti, joka näyttää pohjapiirroksen ja polun
- Tuottaa oma tai soveltaa olemassa olevaa algoritmia parhaan kävelyreitin laskemiseen
- Käyttää kahta edellistä tuotosta onnistuneesti sovelluksessa
- Käyttää sovellusta todellisessa käyttökohteessa

# 2 PROJEKTIN TEOREETTINEN TAUSTA

Graafi on matematiikassa ja tietojenkäsittelyteoriassa käytetty käsite. Graafien avulla voidaan mallintaa ja ratkaista erilaisia ongelmia, esimerkiksi sähköpiirin virtojen laskeminen, kartan värittäminen tai reitin määrittäminen kartalla tiettyjä ehtoja noudattaen. Ongelma, jonka ratkaisusta graafiteorian katsotaan saaneen alkunsa, on *Königsbergin siltaongelma*. Königsbergissä oli seitsemän siltaa, joilla yhdistettiin joen vastarannat ja kaksi saarta toisiinsa. Ongelmana oli sellaisen reitin keksiminen mitä kävelemällä voitaisiin ylittää jokainen silta täsmälleen yhden kerran ja päätyä takaisin samalle maaosalle, josta lähdettiin. /1/

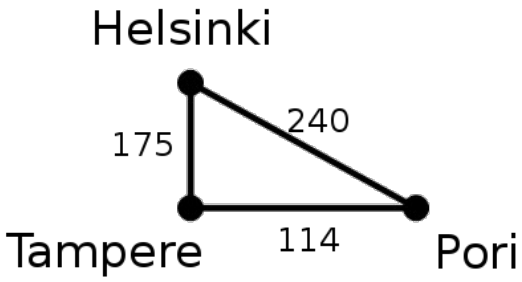
Osia, joista graafi koostuu, nimitetään *solmuiksi* (node, vertex) sekä *kaariksi* (edge). Graafin solmut liittyvät toisiinsa kaarien avulla siten, että jokainen kaari yhdistää tasan kahta solmua. Solmuun voi toisaalta kiinnittyä kuinka monta kaarta vain. Graafi voidaan esittää graafisesti siten, että solmut kuvataan pisteinä ja kaaret pisteitä yhdistävinä viivoina. Useissa tapauksissa sillä ei ole merkitystä, miten pisteet (solmut) järjestellään, kunhan niitä yhdistävät viivat (kaaret) ovat oikein. Viivojen suoruudella tai kaarevuudella ei ole merkitystä. /2/



Kuva 1: Kolme erilaista graafia

Kuvassa 1 on esitetty kolme yksinkertaista graafia. Yksinkertaistuksen vuoksi graafeja, solmuja tai kaaria ei ole yksilöity. Vasemmanpuoleisessa graafissa on kaksi solmua, ja yksi kaari. Keskimmäisessä graafissa on kolme solmua ja kaksi kaarta. Nämä kaksi graafia ovat *yhdistettyjä*, sillä mistä tahansa solmusta pääsee mihin tahansa toiseen solmuun yhtä tai useampaa reittiä. Oikeanpuoleisessa

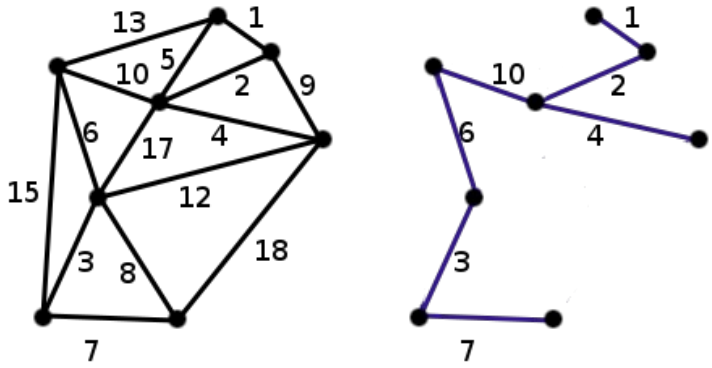
graafissa on neljä solmua ja kolme kaarta siten, että yksi solmu jää ilman kaaria. Graafi ei tällöin ole kytketty. /2/




Kuva 2: Graafi, jossa solmut on nimetty ja kaarille annettu merkitykselliset arvot

Kuvan 2 graafissa esitetään tietoa kolmesta kaupungista ja niiden välisistä etäisyyksistä. Helsingin ja Tampereen välinen etäisyys on 175 kilometriä, Helsingin ja Porin välimatka on 240 kilometriä ja Tampereen ja Porin välimatka on 114 kilometriä. Oletetaan, että henkilö haluaa päästä Porista Helsinkiin. Vaihtoehtoja on kaksi; mennä suoraan Helsinkiin kulkemalla 240 kilometrin matka, tai kulkea Tampereen kautta, jolloin matkan pituudeksi tulee 289 kilometriä. Molemmat ratkaisut ovat hyviä, vaikkakin suora yhteys on selkeästi lyhyempi. Ratkaisuun vaikuttaa muun muassa käytettävä kulkuneuvo (bussi voi pysähdellä matkalla, kevytmoottoripyörällä huippunopeus voi olla 80km/h) ja keliolosuhteet (Helsingin moottoritie on tuiskunnut umpeen, talvirajoitukset tietyillä tieosuuksilla). Kaikki tällaiset tiedot on mahdollista syöttää graafiin, ja halutut tiedot huomioon ottamalla on mahdollista saada ratkaisu.

Lyhimmän kävelyreitin laskeminen oli opinnäytetyön kannalta oleellinen ratkaistava ongelma. Graafiteoriassa samankaltainen ongelmatilanne tunnetaan nimellä *lyhimmän polun virityspuu* (shortest spanning tree eli SPT). Tämä ongelma on jo ratkaistu monella eri tavalla. SPT on kyseessä silloin, kun kaikki graafin solmut on yhdistetty siten, että kaarien *paino* (kustannus, rasite) on pienin mahdollinen. Paino voi olla myös negatiivinen, mutta tällaiset graafit käsitellään yleensä erikseen. Mikäli kaarien painojen summa ei ole pienin, on kyseessä vain yksi monista virityspuista. /3/




Kuva 3: Graafi ja sen virityspuu. Numerot ovat kaarien painoja, solmuja ei ole nimetty.

Graafiteoriaa sovellettiin tämän opinnäytetyön puitteissa kohtalaisen runsaasti. Kaaret vastaavat seiniä tai polkujen suoria osia. Solmut vastaavat nurkkia tai polkujen mutka- tai risteämäkohtia sekä myös hyllypaikkoja ja sijainteja. Kaaren paino lasketaan sen yhdistämän solmujen välimatkana. Tämä on huomionarvoista, koska yleensä solmujen sijainnilla ei ole graafiteoriassa merkitystä – tässä sovelluksessa tilanne on juuri päinvastoin. Paitsi, että tämä vastaa tilannetta käytännössä erittäin hyvin, se poistaa tarpeen asettaa jokaisen kaaren paino käsin.

# 3 ASIAKASOHJELMISTO YLEISESTI

Ohjelmisto, jonka osaa tämä opinnäytetyö käsittelee, tilaaja oli Citidigi Oy. Ohjelmiston ja sen toimintaympäristön kuvaus oli seuraavanlainen:

- Perustoiminnot
  - Myymälän, laivan, kampuksen tai muun vastaavan alueen pohjapiirroksen näyttäminen graafisesti
  - Alueella sijaitsevien kohteiden kuten esimerkiksi myymälässä myytävien tuotteiden, toimistorakennusten konttoreiden tai laivan hyttien listaaminen kategorioittain käyttäjälle

    - Alueella sijaitsevien kohteiden etsiminen vapaalla sanahaulla
    - Optimaalisen reitin laskeminen käyttäjän eli laitteen sijainnista valitun kohteen luokse
    - Reitin näyttäminen käyttäjälle graafisesti pohjapiirrokseen liitettynä
- Lisätoiminnot
    - Mainontaan soveltuva sisällöltään vaihtuva sivuttain rullaava tekstikenttä
    - Helposti muokattavissa oleva ulkoasu tulevien asiakkaiden ulkoasuvaatimuksien kustannustehokasta täyttämistä varten
- Laiteympäristö
    - Pystysuuntaan käännetty kosketusnäytöllinen LCD-televisio kytkettynä tietokoneeseen
    - Käyttöjärjestelmä Windows XP
    - Laitteiston suorituskyky valittaisiin siten, että se suoriutuu ongelmitta ohjelmiston suorittamisesta

Alkupalaverissä sovitun sisällön lisäksi toteutettiin myös seuraavat ominaisuudet:

- Ylläpito-ohjelmisto
    - Karttadatan muokkaus graafisesti
    - Reittidatan muokkaus graafisesti
    - Kohdedatan muokkaus
- Ulkoasueditori
    - Komponenttikohtainen ulkoasun määrittäminen
    - Välitön esikatselutoiminto
- Tietokantaliittymä sekä tietokannan alustustoiminnot

# 4 TYÖKALUT

## 4.1 Qt

Qt on alun perin norjalaisen Trolltech ASA:n kehittämä ohjelmiston kehitys- ja suoritusympäristö (eng. *framework*), joka mahdollistaa alustariippumattoman ohjelmistokehityksen helposti ja nopeasti. Myöhemmin tekstissä tähän sovellusympäristöön viitataan nimellä Qt-alusta. Nokia osti Trolltechin vuonna 2008, jonka jälkeen Qt-alustaa on alettu kehittää voimakkaasti eteenpäin. /4/

Qt-alusta on saatavilla eri lisensointimalleilla. Avoimen lähdekoodin versiota edustaa GPL- sekä LGPL-versiot. GPL-lisensoitu versio sallii vain avointen, ei-kaupallisten ohjelmistojen kehityksen. LGPL-lisensoidulla versiolla on mahdollista tehdä rajatusti suljettua, kaupallista lähdekoodia. Kaupallinen versio sallii täysin suljetun ohjelmiston tekemisen. /5/ Qt-alustaa käytetään monissa yrityksissä ja ohjelmistoissa, niin suljetun kuin avoimen lähdekoodin kautta. /6/

Itse projektin lähdekoodin kirjoittaminen, testaaminen sekä ajaminen tapahtui Qt-alustan omilla kehitystyökaluilla. Qt-alusta osoittautui projektin aikana varsin monipuoliseksi ja tehokkaaksi.

## 4.2 MySQL

MySQL on Oraclen kehittämä tietokantaohjelmisto, joka on maailman suosituin avoimen lähdekoodin tietokanta nopeutensa, luotettavuutensa ja helppokäyttöisyytensä ansiosta /7/. Siitä on saatavilla avoimen lähdekoodin GPL-lisensoitu versio. Kaupallisia käyttötarkoituksia varten on hakittava maksullinen lisenssi. /8/ MySQL valittiin tähän projektiin juuri edellä mainituista syistä.

MySQL-tietokantaa muokattiin enimmäkseen MySQL Query Browser -ohjelmaa käyttämällä. Loppuvaiheessa tutustuttiin myös MySQL Workbench -ohjelmistoon.

# 5 DATAN KÄSITTELY YLEISESTI

## 5.1 Tietokantaliittymästä

Tämä opinnäytetyö käsittelee ohjelmistossa esiintyvää dataa vain siltä osin, kuin sitä käsitellään tietokannan ja tietokantaliittymän ulkopuolella. Tämä tarkoittaa sitä, että tiedon tallentamista varastoon ja sen noutamista sekä näihin liittyviä tarkistus- ja muuntamistoimenpiteitä käsitellään tässä opinnäytetyössä vain pintapuolisesti.

### 5.1.1 Asiakasohjelmisto

Asiakasohjelmisto käsittelee tietokantadataa ns. yksisuuntaisesti. Se siis vain vastaanottaa dataa, eikä tallenna tietokantaan mitään tietoa. Tiedon hakemisen vaiheet voidaan listata seuraavasti:

- Tietokantayhteys muodostetaan. Jos yhteyttä ei voida muodostaa, ohjelman suoritus keskeytyy.
- Tietokannan perusrakenne tarkastetaan. Jos tietokanta ei sisällä ennalta valittuja tauluja, eli tietokanta on väärä tai alustamaton, ohjelman suoritus keskeytyy.
- Tietokannasta noudetaan kaikki ohjelman suorittamiseen tarvittava data.
- Noudettu data käsitellään ja muokataan sellaiseen muotoon, että se soveltuu käytettäväksi ohjelmiston sisällä.
- Dataa käytetään, ja määräajoin tietokannasta tarkistetaan aikaleima. Mikäli päivitys havaitaan, kaikki data noudetaan uudestaan.

### 5.1.2 Ylläpito-ohjelmisto

Ylläpito-ohjelmisto käsittelee dataa monimutkaisemmin verrattuna asiakasohjelmistoon. Se sekä lukee että kirjoittaa tietokantaan, ja se suorittaa eri

oikeellisuustarkistuksia ennen kirjoittamista. Ylläpito-ohjelmiston datan käsittelyä voidaan kuvata yksinkertaistetusti seuraavasti:

- Tietokantayhteys muodostetaan. Jos yhteyttä ei voida muodostaa, ohjelman suoritus keskeytyy.
- Tietokannan perusrakenne tarkastetaan. Jos tietokanta ei sisällä ennalta valittuja tauluja, eli tietokanta on väärä tai alustamaton, ohjelman suoritus keskeytyy.
- Tietokannasta noudetaan kaikki ohjelman suorittamiseen tarvittava data.
- Käyttäjä muokkaa tietoja. Myös muokkausten aikana suoritetaan tarkastuksia datan oikeellisuuden säilyttämiseksi.
- Käyttäjä tallentaa tekemänsä muutokset. Ohjelmisto tarkastaa datan oikeellisuuden perusteellisesti, ja tallettaa tehdyt muutokset tietokantaan. Tämän jälkeen käyttäjä voi jatkaa uusien muokkausten tekemistä.

# 6 POHJAPIIRROS- JA REITTIDATAN RAKENNE

## 6.1 Yleistä

Sovelluksessa käytetyn kartan peruskomponentit ovat pohjapiirrosdata, eli seinät, hyllyt ja muut liikkumattomat rakenteet, sekä reitit, joita pitkin ihmiset opastettaisiin oikeaan paikkaan. Ne ovat toteutuksen tasolla niin lähellä toisiaan, että ne voidaan tässä yhteydessä yleistää karttadataksi.

Karttadata koostuu kahdesta peruselementistä: Pisteet ja polygonit. Pisteellä on seuraavat ominaisuudet:

- Tunniste eli juokseva, uniikki numero
- X- ja y-koordinaatti
- Kerros

Polygonilla on seuraavat ominaisuudet:

- Lista pisteiden tunnisteista, joista polygoni koostuu

- Tyyppi

## 6.2 Pisteen rakenne

Pisteen ominaisuudet ovat verrattain yksinkertaiset: Jokaisella pisteellä on tunniste, koordinaatti ja kerrostieto. Kerrostieto voidaan rinnastaa z-koordinaattiin, mutta kyseessä ei ole kolmiulotteinen tila, vaan pisteet on yksinkertaisesti ryhmitelty kerroksittain. Tämä heijastaa tosielämää konkreettisesti; voihan monikerroksisen talon piirroksen tehdä piirtämällä jokainen kerros omalle paperilleen.

Pistedatan säilyttäminen tietokannassa tapahtuu siten, että jokaisella pisteellä on oma uniikki järjestysnumero (id), jonka avulla piste voidaan yksilöidä, ja siihen voidaan viitata yksikäsitteisesti riippumatta sen koordinaatista ja kerroksesta. Sen (x,y) koordinaatti sekä kerroksen numero, jossa piste sijaitsee, ovat uniikin pisteen ominaisuuksia. Niiden mukaan voidaan suorittaa rajaavaa hakua (esimerkiksi kaikki tietyn kerroksen pisteet, jotka ovat tietyn suorakulmion sisällä), mutta pisteeseen viittaaminen tapahtuu aina tunnisteen perusteella.

Pistedatan säilyttäminen ajoaikana ohjelman muistissa on toteutettu eri tavalla. Pisteet on sijoitettu tietorakenteeseen seuraavasti:

```
QMap < int, QMap < int, QPoint > > points
```

Ulommassa säiliössä on joka kerrosta kohden oma säiliö, joka on indeksoitu kerroksen numeron mukaan. Tämä rakenne on toisaalta niin yksinkertainen, että sen olisi voinut helposti korvata normaalilla taulukolla, mutta tätä muutosta ei enää projektin loppuvaiheessa alettu tehdä, sillä siitä saavutettu suorituskykyparannus olisi ollut käytettyyn työmäärään verrattuna mitätön. Toisaalta tällöin kerrosten luominen ja poistaminen on yksinkertaisempaa perinteiseen taulukkorakenteeseen verrattuna, sillä muistin varaaminen ja vapauttaminen tapahtuu automaattisesti.

Jokaisessa näistä tauluista on edelleen tunnisteen perusteella indeksoitu pisteen koordinaatti. Tällä ratkaisulla on puolensa. Hyvänä puolena voidaan pitää tilanteita, joissa käsitellään vain yhden kerroksen pisteitä. Tällöin voidaan käydä läpi kaikki vain tietyn kerroksen pisteet, jolloin läpikäyntiin kuluva aika on suoraan verrannollinen kerroksen kokoon, ei pisteiden kokonaismäärään.

Epäedullisena tilanteena voidaan mainita tilanne, jolloin tiedetään pisteen tunniste, mutta ei sitä, missä kerroksessa se on. Tällöin täytyy käydä jokainen kerros läpi, kunnes piste joko löytyy, tai pistettä ei ole olemassa. Jos kerroksia on N kappaletta, hakuja tehdään keskimäärin N/2 kappaletta. Säiliön *QMap* sisäinen toteutus hyödyntää tiivistetaulua (eng. *Hash table*). Koska tiivistetaulusta kohteen etsiminen on ajallisesti lähes aina vakio, etenkin sisällön pysyessä optimoinnin jälkeen muuttumattomana, tämä ei aiheuta havaittavaa viivettä yksittäisen pisteen kohdalla.

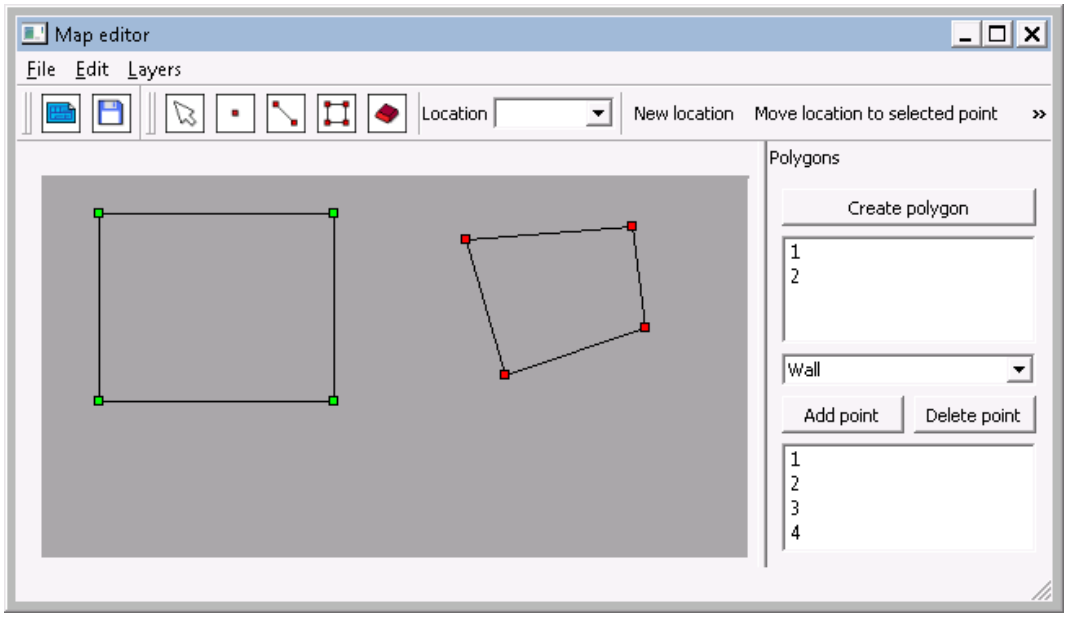
## 6.3 Monikulmioiden rakenne

Seinät, hyllyt ja muut vastaavat pohjapiirroselementit mallinnettiin ohjelmistossa polygoneina. Polygoni on käytännössä sarja pisteitä. Ohjelman puitteissa oli otettava erityisesti huomioon, että yksi polygoni saattoi koostua vain saman kerroksen pisteistä. Tämä aiheutti lukuisia tarkistuksia polygonien käsittelyssä.

```
QMap < int, QMap < int, QList < int > > > polygons
```

Jos polygonin tyypiksi oli merkitty seinä, se tulkittiin ohjelman suorituksen aikana suljetuksi monikulmioksi, joka piirrettiin kartalle halutulla tyylillä.

Kuvassa 4 nähdään kaksi pisteistä muodostuvaa seinää siten, kuin ne graafisesti ylläpito-ohjelmassa esitettiin.




Kuva 4: Karttaeditori, jossa näkyy kaksi seinää kyseisessä kerroksessa.

Seinään viittaaminen, kuten pisteeseenkin viittaaminen, tapahtui viittaamalla ensin haluttuun kerrokseen. Näin päästiin käsiksi listaan kerroksen polygoneista. Tietyssä kerroksessa polygoniin viitataan suoraan sen tunnisteella. Kun polygonien piirtämistä alettiin suunnitella, ensimmäinen pulma oli luonnollisesti se, että tämä tietorakenne ei sisällä tietoa polygonin tyypistä, eli onko se seinä vai reitti. Tämä ongelma päätettiin ratkaista sijoittamalla tämä tieto rinnakkaiseen tietorakenteeseen:

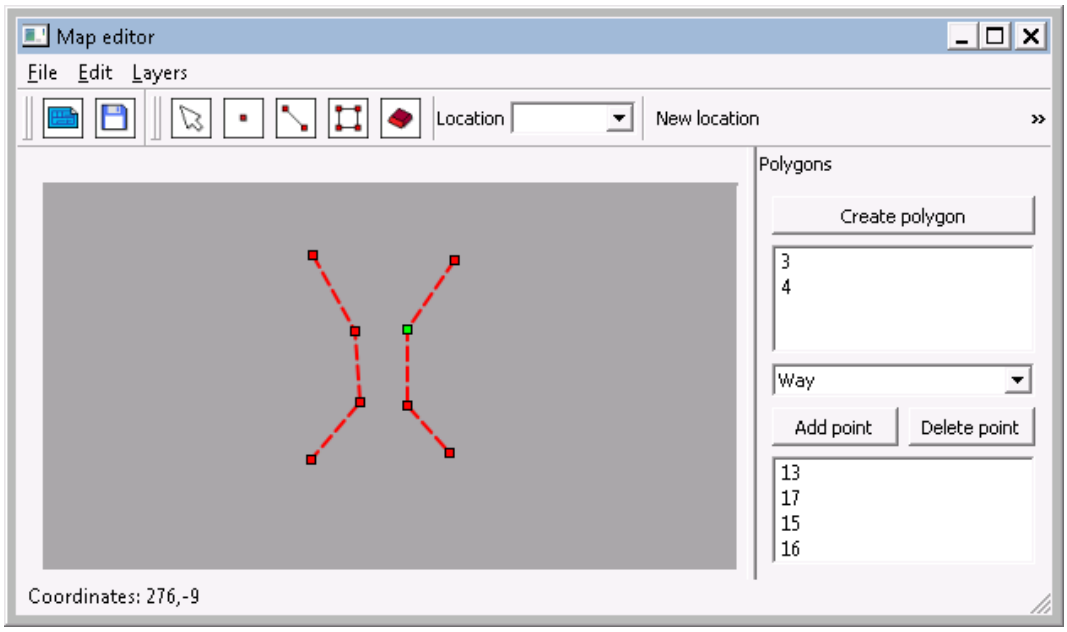
```
QMap < int, QString > polygonTypes
```

Tämä tietorakenne yksinkertaisesti sisälsi polygonin tunnisteen ja sen arvona saatavan tyypin. Näin jokaisen polygonin tyyppi oli saatavilla nopeasti riippumatta siitä, missä kerroksessa se sijaitsi.

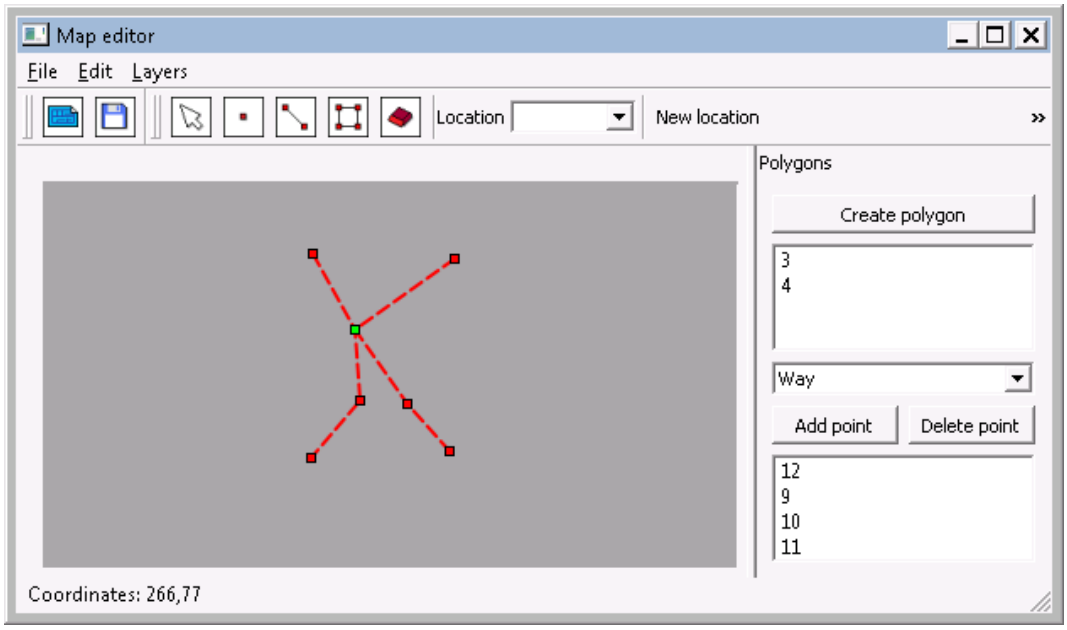
## 6.4 Polkujen rakenne

Polkujen rakenne oli ohjelmistossa hyvin lähellä seinien rakennetta. Siinä missä seinät ja hyllyt olivat ryhmä toisistaan erillisiä polygoneja, reitit koostuivat

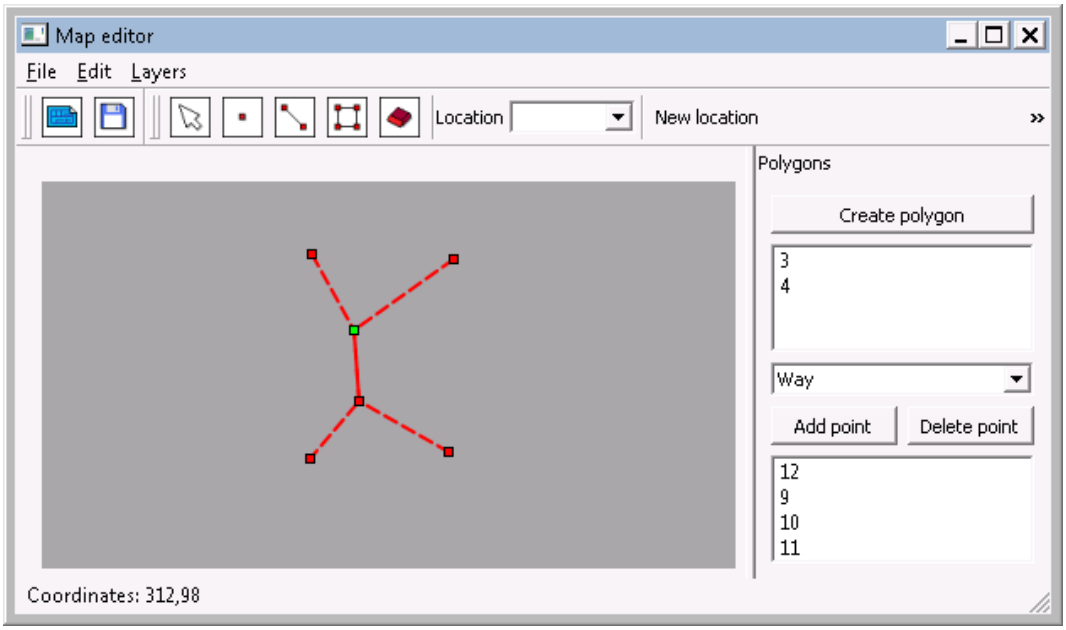
osittain samoista pisteistä. Tämä on luonnollinen vaatimus poluille: Jos yhdeltä reitiltä pääsee toiselle reitille, on näiden reittien vähintään sivuttava toisiaan, ristettävä, tai kuljettava ainakin osan pituudestaan samaa reittiä. Tämä on havainnollistettu kuvissa 5, 6 ja 7:




Kuva 5: Kaksi polkua, joiden välillä ei ole mahdollista siirtyä.




Kuva 6: Kaksi polkua, jotka koskettavat toisiaan. Polulta pääsee siirtymään toiselle polulle.




Kuva 7: Kaksi polkua, jotka kulkevat osittain samaa reittiä.

Polkujen kannalta oli oleellista, että ylläpito-ohjelmassa tallennettu data oli oikeellista. Tämä tarkoitti käytännössä sitä, että mistä vain tietyn kerroksen pisteestä pääsi mihin vain toiseen saman kerroksen pisteeseen.

# 7 KARTTAKOMPONENTIN RAKENNE

## 7.1 Muodostin

Karttakomponentin toteutuksessa kyseisestä luokasta on olemassa ohjelman suorituksen aikana yksi instanssi. Luokan muodostimessa suoritettiin alustustoimintoja, joilla varmistettiin komponentin minimaalinen toiminta. Käytännössä tämä tarkoitti sitä, että boolean-muuttujille asetettiin alkuarvot. Tällä saavutettiin toiminnallinen varmuus siinä mielessä, että vaikka kartan piirtoa yritettäisiinkin epäonnistuneen karttadatan noutamisen jälkeen, boolean-muuttujien arvot estäisivät NULL-tyyppisistä tietorakenteista lukemisen, eikä ohjelma kaatuisi.

Karttakomponentin yhteydessä alustettiin seuraavat boolean-muuttujat:

- initialized
- animationRendered
- receiveDataInProgress

Muuttuja ”initialized” ilmaisee sitä, onko luokan erillinen alustusfunktio jo suoritettu. Funktio voidaan suorittaa vain kerran, ja sen lopuksi muuttujalle asetetaan ”true” arvo. Mikäli funktio yritetään ajaa syystä tai toisesta uudelleen, sen suoritus loppuu heti alkuunsa, eikä ohjelman toiminta häiriinny.

Muuttuja ”animationRendered” ilmaisee sitä, onko senhetkisen polun animaatio jo saatu piirretyksi välimuistiin. Mikäli näin ei ole, paintEvent() funktion toteutuksessa ei polkua vielä piirretä, vaan sitä yritetään uudelleen seuraavalla päivityskerralla. Tähän muuttujaan tehdään muutoksia vain silloin, kun uusi polku on piirretty, ja silloin kun näkymä muutetaan ”pois” reitin piirtämisestä.

Muuttuja ”reveiceDataInProgress” ilmaisee sitä, että tietokannan päivitystoiminnot ovat käynnissä, eikä mihinkään dataan tulisi koskea. Ohjelma hyödyntää vain yhtä säiettä, mutta koska signaalit ja slotit -periaatteen mukaan suoritusvarmuudesta ei ole täyttä varmuutta, tätä muuttujaa käytetään manuaalisesti varmistamaan, ettei kartanpiirtoa yritetä kesken datan muokkaamisen, mikä johtaisi ohjelman kaatumiseen. Mikäli tämän muuttujan arvo on tosi, valtaosa funktioista palautuu heti. Lisäksi eri numeeristen muuttujien, kuten kerroksen ja kohdesijainnin tunniste, asetetaan nolla, joka funktioiden suorituksen aikana saa aikaan piirtotoimintojen ohittamisen.

## 7.2 Alustus

Alustusfunktiota initialize() kutsutaan tasan kerran ohjelman suorittamisen yhteydessä. Tähän mennessä kaikki ohjelman luokat on luotu. Alustusfunktio suorittaa seuraavat toiminnot:

- Komponentin koon laskeminen ruudulla. Tämä riippuu näytön

resoluutiosta.

- Pohjapiirroskuvien luominen oikean kokoisina
- Reitinlaskentaluokan instanssin luominen
- Ruudunpäivitysajastimien alustus
- Tietokantadatan päivitys
- Alustusfunktion tilaksi merkattu suoritettu (boolean-muuttuja initialized)

Tietokannasta haetun datan käsittely tapahtuu kartanpiirtokomponentin sisällä. Projektin loppuvaiheessa huomattiin, ettei tällä toiminnolla ole kartanpiirron kanssa mitään tekemistä, vaan kyseessä on ennemminkin datan päivityksen vaihe. Luokka päätettiin kuitenkin jättää ennalleen, sillä sen korjaaminen olisi aiheuttanut kohtuuttoman määrän lisätyötä saavutettu hyöty huomioon ottaen.

Tiedonkäsittelyn aluksi asetetaan boolean-muuttujalle ”reveiveDataInProgress” arvo ”true”. Tällä saadaan aikaiseksi ”työrauha”, eli ohjelman muut komponentit tietävät, että päivitys on meneillään, eikä tietoihin saa koskea.

Tämän jälkeen tyhjennetään ja täytetään uudelleen sisäiset tietorakenteet seuraavassa järjestyksessä: Sijaintitiedot, pisteet, staattiset sijainnit, laitteen sijainti sekä ID, polygonit ja niiden tyypit. Tämän jälkeen nollataan kerrosten pohjapiirrokset yksinkertaisesti täyttämällä ne läpinäkyvällä värillä. Tämän jälkeen, kuitenkin ennen seinien ja hyllyjen piirtämistä, piirfetään pohjapiirrokseen tietokannasta ohjelman ulkopuolella valmisteltu kuva, mikäli se on olemassa. Tällä saavutetaan parhaimmillaan se, että kartan pohjapiirroksen muutokset voidaan piirtää suoraan ulkoisella sovelluksella, esimerkiksi Adobe Photoshop, ja siirtää vektorimuodossa tietokantaan. Tietokantadatan editointiohjelmassa on toki valmiudet piirtää seinä ja muut kiinteät kohteet paikalleen, mutta tällä saavutetaan se etu, että näkyvien kohteiden piirtäminen voidaan suorittaa ulkoisella ohjelmalla, jolloin työ on nopeampaa ja ulkoasu saadaan näyttävämmäksi. Tällöin ohjelman itsensä vastuulle jää vain reittien piirtäminen.

Tietojenpäivitysfunktion loppuvaiheessa asetetaan reitinlaskentakomponentin

viittausmuuttujat kohdilleen, jotta reitinlaskutilanteessa em. luokalla olisi oikeat tiedot käytettävissään. Lopuksi ruudun näkymä päivitetään.

Seuraavissa kappaleissa eritellään piirtämisen eri vaiheet tarkemmin.

## 7.3 Seinien muodostaminen

Tietokannasta haettu polygoni- ja tyyppidata on vielä käsiteltävä, jotta sitä voitaisiin käyttää ohjelman suorituksen aikana. Tässä yhteydessä käsitellään vain seinä-tyyppisiä polygoneja, mutta vastaava toiminto voidaan tarpeen vaatiessa muuntaa vähäisin muutoksin toimimaan vapaasti valittavissa olevan tyypin käsittelyksi.

Toiminto suoritetaan jokaiselle polygonille. Mikäli sen tyyppi on seinä, toiminto luo uuden polygonin tiettyyn kerrokseen oikealla tunnisteella, ja lisää pisteet kyseiseen polygoniin.

## 7.4 Seinien piirtäminen puskuriin

Seinien piirtäminen on toteutettu ohjelmassa siten, että aluksi mahdolliset kerrosten pohjapiirrokset poistetaan välimuistista. Tämän jälkeen kyseistä funktiota kutsutaan kerran jokaista kerrosta kohti. Tuloksena on jokaisen kerroksen päivittyminen.

Funktion alussa pyydetään väliaikainen lista kerroksessa sijaitsevista seinistä. Tämän jälkeen tarpeen vaatiessa luodaan uudelleen, ja tyhjennetään. Taustalle piirretään tietokannasta mahdollisesti löytyvä kerroskohtainen taustakuva, ja tämän päälle piirretään seinät yksitellen.

Lopuksi taustakuvaan piirretään staattiset kohteet. Staattisia kohteita ovat sellaiset kohteet, joiden halutaan näkyvän aina ruudulla, esimerkiksi uloskäynnit ja WC:t.

## 7.5 Polun piirtäminen puskuriin

Graafisen näkymän tärkeimpiä yksittäisiä osia laitetta käyttävän asiakkaan opastamiseen on ruudulla näkyvä polku. Havainnollisuuden takia polusta päätettiin tehdä animoitu, jotta reitti perille olisi mahdollisimman helppo hahmottaa. Ennen funktion kutsumista reitti kohteeseen on jo olemassa, joten jäljelle tässä yhteydessä jää piirtäminen. Reitinlaskenta käsitellään tarkemmin kappaleessa kahdeksan.

Jotta graafinen suorituskyky olisi maksimaalinen, välimuistiin piirtämisestä on tehty jonkin verran raskaampi. Tämän seurauksena itse ruudulle piirtofunktio on huomattavasti yksinkertaisempi. Aluksi piirretään ensimmäinen kuva animaatiosta. Se tapahtuu piirtämällä kyseisen puskurikuvan päälle kyseisen kerroksen taustakuva. Tämän jälkeen sen päälle piirretään alku- ja päätepisteiden merkkaavat ympyrät ja lopuksi katkoviiva. Lopulta kuvan päälle piirretään staattiset kuvakkeet, jotta ne näkyisivät polun päällä.

Kun ensimmäinen kuva on saatu valmiiksi, muutetaan katkoviivan suhteellista aloituskohtaa. Näin katkoviiva piirretään samalle reitille, mutta katkoviivan aukot ovat hieman eri paikassa. Näin piirretään jokainen kuva erikseen, ja lopulta tuloksena saadaan animaatioiden kuvat.

On mainittava, että animaatiokuvat eivät sisällä pelkkää polkua, vaan täydellisen näkymän koko kerroksesta. Jokainen animaation välimuistikuva sisältää saman informaation, ja ainoa ero näiden kuvien välillä on katkoviivan eteneminen. Tästä seuraa se, että grafiikkaa ruudulle piirrettäessä aluetta ei tarvitse ensin tyhjentää, vaan uusi kuva peittää vanhan kuvan täysin alleen. Näin kuvan tyhjentäminen jää pois ja toiminto nopeutuu.

## 7.6 Grafiikan piirtäminen ruudulle

Grafiikan piirtäminen ruudulle on toteutettu korostetun yksinkertaisesti omassa, funktiossaan nimeltä paintEvent(). Sen toteutus on kolmiosainen:

1) Alusta QPainter-luokka, jonka avulla ruudulle piirto tapahtuu.
2) Mikäli animaatio on piirretty välimuistiin, piirrä kyseinen ruutu pohjapiirroksen päälle, korvaten nykyinen näkymän. Tyhjennystä ei tarvita.
3) Mikäli animaatiota ei ole piirretty, piirrä pelkkä pohjapiirros ruudulle.

Piirtotoiminnot käyttävät funktiota QPainter::drawImage(...), jonka avulla piirretään bittikartta piirrettävään kohteeseen, esimerkiksi juurikin QWidget. Suorituskyvyn maksimoimiseksi bittikartat ovat tarkalleen oikean kokoisia, ja ne piirretään origoon, jolloin esimerkiksi skaalaamisesta aiheutuvaa lisäkuormitusta ei ole.

# 8 REITINLASKENTAKOMPONENTIN RAKENNE

## 8.1 Yleisesti reitinlaskenta-algoritmeista

Parhaan reitin laskenta kahden pisteen välillä ennalta tuntemattomassa kartassa eli graafissa oli tämän projektin suurin yksittäinen haaste. Tämän toiminnon saavuttamiseksi luotiin algoritmi, joka toimi niin sanotusti riittävän hyvin. Tämä tarkoittaa sitä, että sinä aikana, kun ohjelmiston toiminta testattiin, algoritmin viimeisin versio ei tuottanut vääriä tuloksia. Algoritmin testaaminen tapahtui enimmäkseen muiden komponenttien testaamisen yhteydessä, ja polun hyvyyttä pidettiin jatkuvasti silmällä, kun ohjelmaa käytettiin.

Dijkstran algoritmi toimii pääpiirteittäin seuraavasti: /9/

1. Luo lista kaikista pisteistä, joissa ei vielä ole käyty
2. Jokaiselle pisteelle tee seuraava:
    1. Aseta matkaksi pisteeseen ääretön (tai hyvin suuri arvo)
    2. Poista viittaus polun seuraavaan pisteeseen
    3. Lisää piste listaan pisteistä, joissa ei ole vielä vierailtu
3. Aseta lähtöpisteen matkan arvoksi nolla

4. Niin kauan, kuin vierailemattomien pisteiden lista sisältää pisteitä
    1. Poista piste, jossa ollaan, vierailemattomien pisteiden listalta
    2. Jokaiselle naapuruspisteelle, joka on nykyiseen pisteeseen suorassa yhteydessä ja joka on vierailemattomien pisteiden listalla, lasketaan uusi reitin mitta. Mikäli se on pienempi, kuin pisteessä jo oleva reitin mitta, aseta uudeksi parhaaksi naapuriksi piste, jossa ollaan.
5. Aseta seuraavaksi pisteeksi piste, jota ei ole poissuljettu, ja jonka reitin mitta on pienin
6. Jos pisteitä jäljellä, aloita alusta. Jos ei, palauta reitti kohteesta määränpäähän
7. Mikäli polkua kahden annetun pisteen välillä ei ole, tai vierailemattomien pisteiden lista ei ole tyhjä, kuvaaja on virheellinen. Muuten suoritus onnistui.

## 8.2 Käytetty reitinlaskenta-algoritmi

Reitinlaskennassa käytetty itse kehitetty algoritmi on periaatteeltaan seuraavanlainen:

1. Sijoita kohdistin lähtöpisteeseen
2. Valitse pisteen suorista naapureista se, josta on lyhin viivasuora etäisyys kohdepisteeseen
3. Merkkaa piste käsitellyksi
4. Siirry kyseiseen pisteeseen
5. Jos piste ei ole loppupiste, jatka tutkimalla viereiset pisteet kuten edellä.
6. Jos piste on loppupiste, siirrä kursori loppupisteeseen ja aloita jälkimmäinen vaihe
7. Tutki kaikki pisteen naapurit, paitsi parhaaksi merkattu, paremman reitin varalta. Mikäli jonkun muun pisteen kautta reitille palaava polun mitta on lyhyempi, kuin jo olemassa oleva laskettu polku, korvaa reitti uudella reitillä ja jatka vaiheesta kaksi.
8. Kun loppupiste on saavutettu, on reitti valmis.

# 9 TESTAUS

Ohjelmiston testausta suoritettiin jatkuvasti ohjelmoinnin aikana. Erityistä huomiota kiinnitettiin poikkeustilanteisiin jo koodauksen, osittain myös suunnittelun, aikana.

# 10 PROJEKTIN ARVIOINTI

## 10.1 Tavoitteiden täyttyminen

Opinnäytetyön tavoitteet olivat seuraavat:

- Tuottaa komponentti, joka näyttää pohjapiirroksen ja polun
- Tuottaa oma tai soveltaa olemassa olevaa algoritmia parhaan kävelyreitin laskemiseen
- Käyttää kahta edellistä tuotosta onnistuneesti sovelluksessa
- Käyttää sovellusta todellisessa käyttökohteessa

Voidaan todeta, että kolme ensimmäistä tavoitetta onnistuttiin täyttämään. Tarvittavat komponentit toteutettiin opinnäytetyön aikana ja niitä käytettiin osana lopullista ohjelmistoa. Neljäs tavoite, eli tuotteen saaminen loppukäyttöön, ei toteutunut. Syyt tälle olivat projektin toteuttajista riippumattomat.

## 10.2 Jatkokehitys

Projektin jälkeen kävi selväksi, että ainoa perusteltu vaihtoehto toteuttaa reitinlaskenta olisi ollut Dijkstran algoritmi, mutta sitä ei enää näin myöhäisessä vaiheessa nähty järkeväksi implementoida, koska lopputuotteen potentiaalinen asiakas oli jo hylännyt tuotteen hankinnan. Mikäli tuotteelle lopulta löytyy lopullinen tilaaja ja käyttäjä, algoritmin implementointi on tärkeimpiä kehityskohteita. Mikäli nykyinen algoritmi tuottaa myöhemmin ei-optimaalisia tuoloksia, Dijkstran algoritmi implementoidaan ennen sitä. Dijkstran algoritmi soveltuu tähän käyttötarkoitukseen erityisen hyvin, sillä sen tuottama data voidaan tallentaa välimuistiin, ja tämän jälkeen jokaisesta kerroksen pisteestä on tiedossa polku haluttuun kiintopisteeseen (esimerkiksi hissi tai Opastaja-laite) ja tämä mahdollistaa ohjelman viiveettömän toiminnan.

Muita kehityskohteita ovat teemojen ja ulkoasun muokattavuus. Projektin alkuvaiheessa piti olla selvää, että ohjelmistoa tultaisiin käyttämään vain pystysuuntaan käännetyn kosketusnäytön avulla, mutta myöhemmin esille tuli myös asennuskohde, jossa kosketusnäyttö soveltuisi käytettäväksi paremmin vaakatasossa. Tämäkin asiakas hylkäsi projektin ennen kuin koodia ehdittiin mukauttaa vaakasuuntaiseen asennukseen, joten ominaisuutta ei lopulta toteutettu.

# VIITTEET


1. Köningsbergin siltaongelma. [Viitattu 6.12.2010] Saatavilla: http://fi.wikipedia.org/wiki/Königsbergin_siltaongelma
2. Reinhard, D. Graph theory. 3rd ed. Birkhäuser, 2006. 410 p.
3. Cormen, T. H. Introduction to algorithms. 2nd ed. MIT Press, 2001. 1180 p.
4. The Nokia acquisition. [Viitattu 14.11.2010] Saatavilla: http://qt.nokia.com/about/the-nokia-acquisition/
5. Qt Licensing. [Viitattu 14.11.2010] Saatavilla: http://qt.nokia.com/products/licensing/licensing
6. Software that uses Qt. [Viitattu 14.11.2010] Saatavilla: http://en.wikipedia.org/wiki/Category:Software_that_uses_Qt
7. Why MySQL? [Viitattu 14.11.2010] Saatavilla: http://www.mysql.com/why-mysql/
8. MySQL Downloads. [Viitattu 14.11.2010] Saatavilla: http://www.mysql.com/downloads/
9. Wikipedia. Dijkstran algoritmi. [Viitattu 14.11.2010] Saatavilla: http://fi.wikipedia.org/wiki/Dijkstran_algoritmi