Master Thesis
Software Engineering
Thesis no: MSE-2008:07

Intelligent Tourist Information System

Krzysztof Jeleń

Department of
Interaction and System Design
School of Engineering
Blekinge Institute of Technology
Box 520

Contact Information:

Author(s):

Krzysztof Jeleń

E-mail: krzysztof.jelen@gmail.com

External advisor(s):

dr inż. Andrzej Gawrych-Żukowski

Institute of Applied Informatics

Faculty of Computer Science and Management

Wrocław University of Technology

University advisor(s):

Bengt Aspvall Ph.D.

School of Engineering

Department of

Blekinge Institute of Technology

Box 520

SE – 372 25 Ronneby

Sweden

Internet	: www.bth.se/tek
Phone	: +46 457 38 50 00
Fax	: + 46 457 102 45

STRESZCZENIE

W dzisiejszych czasach urządzenia mobilne stały się bardzo popularne. Większość z nas posiada telefon komórkowy, czy też palmtopa. Najczęściej te urządzenia używane są do zaspokojenia potrzeby bycia w kontakcie. Coraz bardziej popularne stają się jednak urządzenia wielofunkcyjne oferujące coś więcej. Takim przykładem są najnowsze telefony komórkowe – smartfony, które oprócz podstawowych funkcji rozmowy oferują także obsługę nawigacji satelitarnej. Urządzenia tego typu umożliwiają podróżowanie bez potrzeby znania dróg, po których się poruszamy oraz podwyższają nasz komfort psychiczny informując na bieżąco gdzie się znajdujemy. Taki system jest w stanie wyznaczyć najkrótszą bądź też najszybszą trasę do zadanego miejsca oraz poprowadzić nas tak, aby bezpiecznie i szybko dojechać do celu. Cechą wspólną tych systemów jest to, że musimy dokładnie wiedzieć gdzie chcemy się udać. Może to dziwić, ale jest rzeczą naturalną, że gdy znajdujemy się w obcym sobie mieście, w obcym kraju nie koniecznie musimy wiedzieć, co jest warte zobaczenia i gdzie powinniśmy się udać.

W niniejszej pracy przedstawiam inne spojrzenie na problem nawigacji satelitarnej oraz planowania tras. Docelowym użytkownikiem systemu jest turysta, dla którego opracowana aplikacja ma stać się wirtualnym przewodnikiem. Taki inteligentny doradca na podstawie kryteriów użytkownika zaproponuje trasę wycieczki po najciekawszych obiektach w danym mieście. Użytkownik musi tylko określić ile czasu może poświęcić na zwiedzanie oraz jakim środkiem transportu będzie się poruszał.. Ma także możliwość określenia, jakiego typu obiekty chce odwiedzić (np.: muzea, galerie, parki). Na podstawie tak podanych kryteriów system powinien wyszukać i zaproponować kilka tras wycieczek. Trasy wyznaczone przez system nie są najkrótszymi ścieżkami jednak do pewnego stopnia są zoptymalizowane, są „wystarczająco dobre". Wynika to z faktu, że głównym wymaganiem, co do systemu była jego szybkość działania.

Niniejsza praca zawiera także opis procesu tworzenia mapy na potrzeby takiego systemu, oraz zawiera krótką charakterystykę aplikacji, która została napisana, aby to umożliwić. Praca porusza także zagadnienia z zakresu urządzeń mobilnych, systemów operacyjnych dla tych urządzeń oraz zawiera podstawowe informacje na temat nawigacji satelitarnej. Krótkie porównanie istniejących systemów nawigacji satelitarnej wraz z opisem ich możliwości oraz krótką charakterystyką jest również zawarte w pracy.

ABSTRACT

Nowadays people use mobile phones and other mobile devices. Most of us have a small computing device that is always with us. People use it example for calling, as calendar and organizer. Mobile devices with GPS receiver are also used to find paths in navigation. The main disadvantage of those systems is that we have to know places which we want to visit and they usually do not store any usable, valuable information about points of interest except phone numbers and addresses.

The main idea of this thesis was to design a system that will run on most of phones and palms and will be helpful when visiting some new places and cities. This system should be able to find a route using user criteria.

Those criteria should be simple and natural, like for example: a list of museums, the most famous historical objects, restaurants to visit, constraints to travel by bus and by walking. The system should find a path that fulfils those criteria, show it on screen, show names of objects, some short descriptions and photos of them and possible entrance costs. It should also be able to estimate time needed to travel from one object to the next and if it is possible, advise which bus line or other public means of transport may be used. It should be helpful for people that want to visit a city without having much information about it. Paths that are output of this system are only a proposition for trip. They will not be optimal but can not be counterintuitive and must be acceptable by the user.

Keywords: mobile devices, tourism and navigation, path finding, creating maps process.

CONTENTS

1 INTRODUCTION

People travel a lot. Most of us have a mobile phone or a navigation device. If they want to make a successful trip they must prepare for it. At beginning a city which will be visited should be chosen. After that those places, buildings and monuments which are important and will be seen need to be selected to include in the trip. Very important is also calculation about the trip cost and the time needed for it. Sometimes people do not have enough time to prepare themselves or they spend few hours in some city without planning this before. Another fact is that people do not have enough time for planning. Most of them ask friends or go for trips organized by tourist companies where a professional guide is involved. It will be very helpful if a system that provides all information needed to visit a city is available. This system should gather data that are presented in brochures, tourist guides and on web pages. A search mechanism and path finding feature are also one of the requirements for this system. It can be difficult for some persons to find paths that allow visiting particular places, but if these persons have some proposals for trip an appropriate system will be very helpful for them in my opinion. With users having some suggestion is a good base to start organizing and optimizing trips and it is easier to change something than make it from beginning.

I designed an application that fulfils most of those needs. It provide an information like maps of a city, parking places, points of interest (museums, art galleries, restaurants, monuments and similar) with their description, costs of entrance, time needed to visit and at least one photo of these places. The system is also able to find path using user criteria and to display them. This is only a proposition for users and it can be used as something for beginning while planning. The user will get few propositions of trips. He can display the path on map and read about objects which are included in each trip. Every path has calculated length and time. It is not optimal and probably the user changes something but as it was said it is only a proposition and some base plan for the trip.

This thesis concentrates on the system that was designed. It also determines which of its aspects were implemented, which should be added in near future or commercial version and which are difficult to implement. To better understand environment, possibilities and requirements hardware and software aspects of mobile devices must be taken into consideration.

This document is organized into 7 chapters. The next chapter presents three existing path finding systems. Basic functionalities and a scope of those systems are presented in second chapter. The third chapter is an introduction to mobile devices. It presents hardware restrictions which must be taken into consideration when building such systems. It contains information about processors and memory of mobile devices and also mentions navigation systems. In that chapter some basic information about current mobile operating systems are presented. It is a background for the rest work. It shows problems that have to be solved by mobile platform programmers while developing software. The fourth chapter refers about goals of this master thesis in details and presents a proposed solution for planning and organizing trips. It consists from two parts: description of how to select objects using some criteria and, secondly, how to find the best path to visit all of them in specified time. At the end future work and conclusions are presented.

1.1 Background

When visiting a city that you do not know, the first place that will be visited is probably tourist information. We can get maps and obtain information about museums, galleries, tourist attractions, and probably entrance costs. Most of them we can find on the web page designed for this place, but often it is provided in national language only that tourists possibly do not know. Sometimes information is available in English but still this language is not understandable for everybody. Then we must print them to have a hard copy during the journey. Today we have GPS (*Global Positioning System*) receivers and navigation programs which are very useful but none of them can prepare a travel plan if we feed them with information like this "I want to visit famous museums in this city and I want to travel by bus, but if it's possible I would like to take a bike trip. I want to have a dinner at noon and taste some local delicacies". If we want to have trip as mentioned, we have to know exactly what we want to see, put addresses in GPS receiver or find them on a map and them find ways to reach them. The question is could we help tourists and give them a tool that is helpful and functional, easy to use and works on numerous portable devices like cell phones, palmtops, etc.

1.2 Aim and objectives

- performing a feasibility study answering the question what are the capabilities and bottlenecks of portable intelligent devices

- modelling the presentation layer and system data flow especially designed for intelligent devices, taking advantage of their specific features but considering their restrictions as well

- design efficient, robust search algorithms with multidimensional optimization

- capabilities from the user point of view – building a localized geographical database of places worth seeing and visiting that will be a back-end of modern smart devices multimedia guiding system integration with GPS receiver (finding way, localization) and GPRS (General Packet Radio Service) receiver (download maps, latest information's) technology

1.3 Research questions

1. Is it possible to build such system and to get satisfying results?

2. What are weaknesses of existing GPS systems and applications for path finding?

3. What are user expectations to a tourist information system? Can they be fulfilled using existing devices?

4. Which algorithm for path-finding will be the best to fulfill user requirements and needs while travelling (different criteria, maybe two or more algorithms should be used for best search results)?

5. How to organize data for most optimal memory consumption?

1.4 Expected outcomes

- Algorithm for finding the path that fulfill users requirements

- Method for data gathering and storage
- Standalone application for portable device that implement above algorithm

1.5 Research methodology

- Case study – map building process, important aspects of creating vector maps, data gathering and storing.
- Experiment – prototype application was tried by few of my friends and people that I know to better understand user needs and fulfill their requirements. Prototype was created using my own experiences with some path finding systems and from my trips.
- Case study – existing algorithms for path-finding, the best will be chosen.

2 EXISTING PATH FINDING SYSTEMS

The idea for intelligent tourist information system is not something new. There are available applications that allow find a path and can be used in mobile devices. Destination point must be exactly known by user to find path. There is no system which can suggest some path that will for example allow visit city without knowing most interesting objects in it. In this chapter I'm describing existing path finding system, I try to show their main advantages and disadvantages. I also try to explain why the system that I want to design is different than existing systems by showing the main gap between them.

2.1 AutoMapa

AutoMapa is a Polish program and it covers Poland only. It has the most precise map of roads and that's the main advantage of this software. The second one is that AutoMapa allows users adding new POIs (Point of Interest) and grouping them. There are some web pages available where users of this system can find some extra information and import it to the program. This can make POI database very huge and complex source of information about important places. It allows having very current localizations of restaurants, hostels and so on. A main disadvantage of AutoMapa is that sometimes it is very slow. Drawing map with many POIs showed on them can take a lot of time. Sometimes AutoMapa have problems with finding path when we are moving in opposite side (we are out of path, AutoMapa estimate path and after that check that we are still out of estimated path and start estimating from beginning). It also do have a very simple mechanism that allow only to search POIs database using name and type and do not allow to add descriptions to each object.

2.2 IGO

IGO cover whole Europe and have very nice user interface. It has a big base of POIs (the museums, the restaurants, the shops and others) but the description for them is rather weak. IGO POI database store only a name and an address of the object without any information that allow to take decision if this object should be seen or it will not be interesting.

It is good software that is working fine and fast. A main advantage of IGO is speed control. Application will play special sound when vehicle is moving over the current limit.

IGO has also a very good and precisely map of most European cities (I test it in Copenhagen, Karlskrona, Malmö, Stockholm, Wrocław, Nowy Sącz). IGO has a lot of language versions.

2.3 TomTom

TomTom is the most popular navigation application on the market. It is the best sold software in this category and it is evolved since 1991. Like IGO it has a huge base of POIs and nice user interface with 3D view on road (the fastest real time map drawing). TomTom is the fastest software for navigation that I used. The user interface can be sometimes confusing, but after few whiles of using it becomes understandable.

TomTom like other two above systems do not provide descriptions about POI that allow selecting objects that are really interesting and should be visited. It only store information like address without photo and open hours.

2.4 Gaps in existing systems

Table 1 presents some short summary about used GPS systems. Opinions presented there are only author's subjective feelings while using those applications.

Table 1 Application summary

Application	Interface	Performance	Covering
AutoMapa	natural	good	only Poland
IGO	natural and limpid	good	depend from version (Europe)
TomTom	natural	very good	

A main gap in those kinds of systems in my opinion is POI database. It is huge and has a lot of objects from different categories. I think that it will be more usable when it allows knowing something more about the points. Users want to know what they can see in a specific place, what are opening hours and cost of entrance. Those and more information allow taking a decision if some place is attractive and interesting and should be visited.

The second gap is that those applications are created for persons that want to travel from point A to B. They need to exactly know where they want to go. In my opinion future for those systems is not only using exactly places that user input to device but also his preferences. Some of as like historical museums, others like art galleries, and some like botanic gardens. The system should ask about that information and using these criteria should build some path. The solution should be provided in short time period and path should be acceptable by user (without loops and choosing shortest path between points). On the other hand it is not good trying to be more intelligent than user but my idea is to help tourists. They should spend time on visiting and learning something about place where they are without wasting time on finding interested objects and planning a tour.

Those three programs allow estimating actual position find path to selected target (using user criteria like shortest, fastest, bike) and they are very helpful for tourists. All of them have in my opinion one gap that I want to eliminate. They are designed for users that want to move from one place to another. They do not allow for example estimate path using criteria like "find shortest path that allow visiting all museums in selected city". From this point of view they are not very useful for people that do not prepare to the trip and do not know what they should visit and which object are available. All of them are very good and they are few years on market and have strong positions, but I think that there is still a place for software that I want to design. Application that will be easy in use, will have a simple path finding criteria (not exactly objects but groups or by description) and give user information about POIs like the entrance cost, a short description, a localization and a photo.

3 Mobile Devices

3.1 Hardware requirements and restrictions

Hardware requirements are very important in mobile devices' world. They do not have a lot of memory and a fast enough processor. In this case applications that are designed for them need to characterize small memory consumption. The phones have from something about 4MB to 32MB build-in memory, the palms have more and actual standard is 32MB or more. In most cases this memory can be expanded by memory cards, but to be more flexible an application should fit in 1 or 2 MB for a phone and 10MB for a palm.

Slow processor need from developers find the best and the simplest algorithm that provide solution on time without big delay. The other problem is that the phone should be always ready to handle the incoming call or the message and the application should never get whole processor time.

The other very important hardware requirement is the screen. It is usually small and any graphic application should have been very carefully designed to not waste the space and display only usable information and be functionally. The navigation system is also connected with the screen. Touch screen are available in devices with bigger displays. They can be handled as typical cursors. In this case user can click on any point of map. Then application should display some information about it and if it is possible show list of object that are near the clicked point. When the device does not have touch screen the user can interact with the application only by keys, and because phones have only few the user interface should be designed including this restriction.

3.1.1 Positioning systems overview

The need of knowing geographical position is older than human kind. In the last several centuries people use compasses, stars and so on. Those methods need simple instruments, calculations and are rather slow. Satellite navigation is the easiest solution for knowing place where we exactly are. Today everybody can buy receiver that allow determine its position. The most popular system is GPS but there are other that can be popular in near future (it is controlled by U.S. army and its accuracy can be manipulated). This chapter is about GPS because estimating position is very important in system that should aid tourists. The other fact is that GPS receivers are rather cheap and there are a lot of mobile devices (palms, some phones) which have them and in near future most of them will have. It does not mean that without navigation system tourist lost but it can be useful and helpful and allow feeling more comfortable.

I also mention about inertial navigation because it is more precise on short distances and it can be used when GPS signal is unavailable [5].

3.1.1.1 GPS

Global Positioning System is a satellite-based navigation system. It was developed by U.S. Department of Defence as a part of NAVSTAR satellite program. It is composed from three segments: space, control and user segments [5].

The space segment consists of 24 satellites with are placed on six circular orbital planes (4 satellites per one orbit). Each orbit is inclined at an angle of 55° relative to equator and they are separated by 60°. Satellites are at an altitude of approximately 20,200km. It is needed to

ensure that at least three satellites will be visible from most places on earth's surface at any time. Each satellite needs 11 hours and 56 minutes to circle the earth [5].
The control segment is responsible for synchronizing satellite's atomic clocks and adjusts the ephemeris of each satellite's internal orbital model. It is managed by U.S. Air Force.
User segment is designed for typical users that want to use GPS receivers to estimate their position. GPS from beginning was designed for use for everybody and it has implemented few techniques that can be used to degrade accuracy of GPS for civilian (non-U.S. military) users. Since 1st may of 2000 United States president forbid to cause interference for civilian users [10]. From this moment accuracy of GPS is estimated as 4-12m [10] and it becomes more popular.

In all systems like GPS position is estimated using data received from satellites. GPS receiver calculates distance to satellites and using theirs position calculate its own. To know the distance to satellites receiver must have very precise clock and satellites need also. This is the main reason why each satellite has an atomic clock and all those clocks must be synchronized. In this system the weakest part is the GPS receiver because it has only the quartz clock and need time synchronization (it is making automatically by receiver using data from satellites) [5].

Nowadays GPS receivers are able to receive from 12 to 20 satellite signals and use them to estimate the position (theoretically signal from only 3 satellites is needed to estimate position with 3mm accuracy) [10].

3.1.1.2 GLONASS

GLONASS (*Globalnaja Nawigacionnaja Sputnikowaja Sistiema*) was developed by Soviet Union and works on this system starts in 1976. Like GPS it is based on 24 satellites but the angle between orbits and equator is 64.8°. Originally it was designed to have 21 operational satellites and 3 in reserve. They should be placed on 3 orbital planes separated by 120° angle. Currently Russia has only 11 satellites, and system should be fully operational in 2010 (24 satellites will be at space) [6].

3.1.1.3 Galileo

Galileo is a European Satellite Navigation System and it is in construction phase. It should be available in 2010. In difference to GPS and GLONASS, Galileo will be controlled by non-military organizations. It will consist of 30 satellites at 3 orbital planes at 56° angle between orbit and equator. Its accuracy for civilian is estimated as 15m horizontally and 35m vertically [9].

3.1.1.4 Inertial navigation

Inertial navigation is based on Newton's first law. Inertial navigations systems are build with two parts: inertial measurement unit and navigation computer.
Inertial navigation unit must have two components accelerometer – measure acceleration and gyroscope – measure rotation. Each sensor measure changes only in one dimension. They need to be barracked together, three for each unit. The navigation computer calculates the output data from measurements units and estimates the position [5].

Inertial navigation in most of cases has a military application and it was originally developed to navigate rockets. Today it is applicable with GPS receiver and it can be used to increase accuracy of GPS systems – this method is called Kalman filtering.

3.2 Software requirements

Mobile devices are like small computers. A main difference is that they should have a special feature – they should work without crashes. Most users do not want to reboot their phone after few hours of use. Device should also be able to handle several actions like incoming call or message independent from actually working applications (real time aspects of mobile phones). All those features can be implemented from scratch for each phone but it increases costs, and strongly ties the software to a specific phone. The solution is to use an operating system that have all those functionalities and provide some API that is independent from the hardware and manufacturer.

I choose Windows Mobile and Symbian as platforms for my software. Those operating systems are the most popular mobile operating systems on the market. Symbian OS is most popular operating system for phones and it is very easy to buy phone with this OS. Windows Mobile almost does not have a competitor because only two or three models of palms are powered by Linux. At next paragraphs I describe those operating systems and compare them to other alternative operating systems that are available on the market and can be run on mobile devices like cellular phones.

In fact the system was designed only for Windows Mobile devices. Symbian OS version is still in developing phase and it is not completed.

3.2.1 Symbian OS

Symbian derive from Psion Company and their EPOC system. It was born in 1998 when phone manufactures were looking for some operating system. Today Symbian OS is the most popular operating system for smartphones (it is full functional phone with personal computer like functionality). The Symbian consortium is owned by Nokia, Panasonic, Psion, Samsung and Sony Ericsson [2].

From the beginning Symbian OS was designed for communication and for mobile devices. It is characterized by low memory consumption. It can be run on different platforms with different CPU that are compatible with ARM architecture RISC processors. Symbian also use an MMU unit to improve speed and to protect memory [7].

Symbian OS is able to handle different memory types (ROM, RAM). System image is stored in ROM memory and allows to executing-in-place (executed code is not copied to RAM but processor can read instruction from place where code is written as a file). It also can be copied to RAM to improve performance. Copying applications to memory needs more memory resources. Because of that only critical libraries are market to execute in RAM [7].

Like other contemporary operating systems Symbian OS also is basing on DLL and allow third part applications to invoke functions from them. Symbian use a special type of DLLs (Dynamic Linked Library) called polymorphic DLL. They are like interfaces and will be described later [7].

The Symbian OS core is very small, something about ~200kB. It supports multi threading. Peripheral hardware is handled by special DLL's that provide all functionality of external devices (like keyboard, camera). Such kind of libraries is running in privilege mode, so communication with devices need to switch momentarily to this mode – context of application is switched to kernel, MMU is able to handle that without flushing the cache. Those kinds of libraries are created from two parts: one is running in privilege mode, and the second one in user mode – it provides an API for the applications.

Symbian OS can be divided for three main platforms:

- Series 60
- Series 80

- UIQ

Each of them has different characteristic and was designed to allow the fulfilment of different user needs. Series 60 is for low cost phones with small displays (Nokia 7650, 6600). Series 80 is used in bigger phones, with half-VGA screen size, Nokia Communicator phone series is based on this version of Symbian OS. UIQ phones are characterized also by a big, touch screen, and in most of cases they do not have a keyboard.

To breakdown system structure few points' of view can be used, but the main parts of Symbian OS are [2]:

- kernel
- basic libraries
- application services, engines and protocols
- application framework
- communication architecture
- middleware feature libraries

Each of above parts will be described below.

▪ Kernel is the unique process that is run in most of operating systems. It manages entire system memory and schedules programs. Only the kernel is allowed to run in privileged CPU mode. If some of third part applications need this access they can be run as kernel plug-ins.

▪ Basic libraries provide basic functionality like string manipulation, access to files and others. They are also used by kernel and provide some kernel functionalities (like creating threads and timers).

▪ Application services, engines and protocols are very similar to basic libraries but they provide access to some more abstract layers in Symbian OS, like for example the contact database.

▪ Application framework is core base for third part applications, handle most generic events in GUI application Communication architecture is responsible for communication via TCP/IP, Bluetooth, IrDA, and also with GSM network (SMS, MMS).

▪ Middleware feature libraries are the rest of Symbian OS functionality that is not covered by previous parts. Multimedia, security and other things placed in this part of Symbian OS.

Figure 1 shows Symbian OS structure.

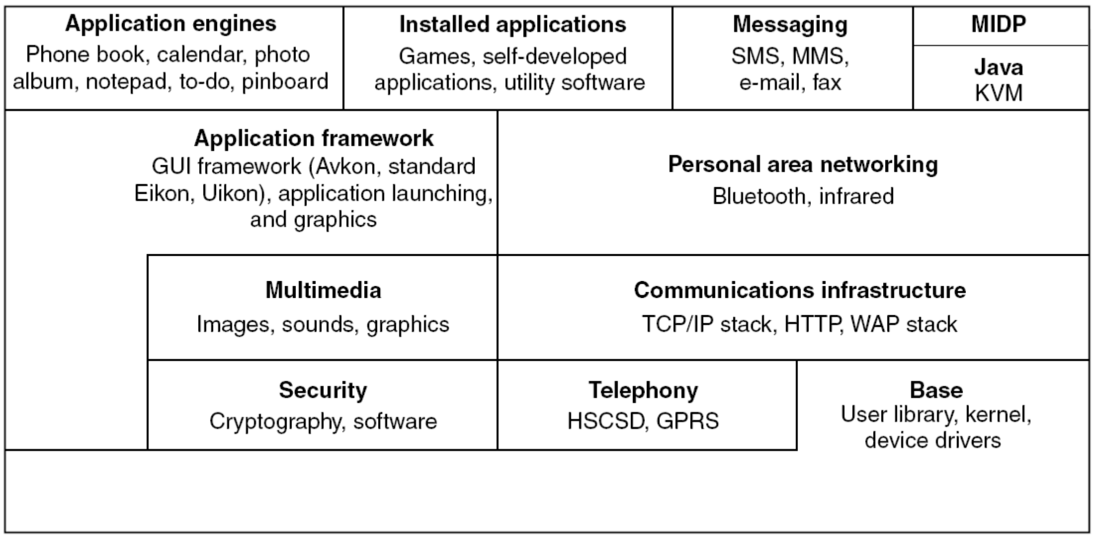

Application engines
Phone book, calendar, photo album, notepad, to-do, pinboard
Installed applications
Games, self-developed applications, utility software
Messaging
SMS, MMS, e-mail, fax
MIDP
Java
KVM
Application framework
GUI framework (Avkon, standard Eikon, Uikon), application launching, and graphics
Personal area networking
Bluetooth, infrared
Multimedia
Images, sounds, graphics
Communications infrastructure
TCP/IP stack, HTTP, WAP stack
Security
Cryptography, software
Telephony
HSCSD, GPRS
Base
User library, kernel, device drivers

Figure 1. Structure of Symbian OS v6.1 for Series 60 [4]

3.2.2 Windows Mobile

Windows Mobile is a compact operating system for embedded and mobile devices based on the Microsoft Win32 API. Windows Mobile is based on Windows CE kernel which was designed for embedded devices and was created as a real-time operating system. Embedded system is a computer that can be used as for example mp3 player or as a controller of traffic lights or as a device which controls some industrial process. Embedded devices very often have a time constraints that mean they must execute actions in time restrictions. Windows Mobile is very similar to the desktop version of Windows and can be run on many different devices: Smartphones, Pocket PCs, and Portable Media Centers. Default features available in all versions of Windows Mobile are described below.

- Today Screen – displays current date, owner information, upcoming appointments, e-mail messages, and tasks and a status bar that display status of Bluetooth and if device has WiFi shows signal strength. This screen often contains icons of applications that are run in background.

- Taskbar – displays current time, allows changing volume, display wireless connection status and like in typical desktop version of Windows it contains a start button.

- Microsoft Office applications are also a part of Windows Mobile. Devices contain programs such a Pocket Word and Pocket Excel and they are very similar to desktop versions but have smaller functionality.

- Windows Mobile provide other typical applications like: Outlook mobile which contains tasks, calendar and contacts; Windows Media Player that plays movies and music files; VPN client

There are a few different versions of this system:

- **Pocket PC 2002**, previous version Pocket PC 2000, both are based on Windows CE 3.0, it was designed for Palm PC and Smartphones

- **Windows Mobile 2003** – released on June 23, 2003, powered by Windows CE 4.2. Designed especially for single-handle use, to devices that don't have touch screen. In March 2004 a new version of this edition was presented as Windows Mobile 2003 SE. It have some improvements and allow to use landscape or portrait screen layout, support VGA resolution (640x480), designed for devices with hardware keyboard.

- **Windows Mobile 5.0** – the latest release of Windows Mobile, originally codenamed "Magneto", was presented in May 2005, powered by Windows CE 5.0 and support .NET Compact Framework with SP2. It has many advantages in user interface, and new features like installed software for GPS receiver management and Office Applications. The main advantage is improved memory management. This version of Windows support Persistent Storage, it means that like in previous versions of Windows Mobile volatile and non-volatile memory are used but in contrast to them only non-volatile memory (flash) is used to store files, and programs data. RAM memory is only used as operating memory (to storing application code and data at runtime), because of that battery supply can be turn of and user does not lose any data. This also increases battery life (something about 50% longer life time, in previous version 50% of battery was always reserved as RAM emergency supply. If the device was not connected to extern power it loses some data).

Windows Mobile fully supports multitasking and allows running applications in background. It also has some real-time aspects inherited from Windows CE that is the base for Windows Mobile systems.

Figure 2 shows a typical screen shot from Windows Mobile.

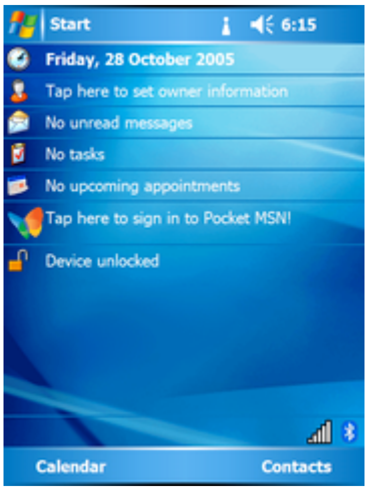


Figure 2. A typical Windows Mobile 5.0 for Pocket PC Today Screen [11]

3.2.3 Other operating systems

On the market other operating system that can be used for mobile devices exists. They are not so popular like system that I mention before but some of companies use them in their devices. Some of them have interesting features and maybe in close feature they will be more popular so they should be taken into consideration.

3.2.3.1 Palm OS

Palm OS is the first operating system for mobile devices. The first time it was used in 1996, in 3Com palmtop called Pilot 1000. The first version works on Motorola processor, latest versions works on ARM processors. It is the simplest operating system for pocket devices. It was designed for maximum usability and functionality with minimal memory and processor time consumption. It can be run on devices with monochromatic and colour displays, up to 640x960 resolution [12]. It is a simple operating system without multitasking since to version 5.0. It has build in few applications like: address book, clock, note pad, sync, memo viewer. Most devices have also installed Grafitti, special software that allow input data using pencil and hand writing (default way is keyboard displayed on a touch screen) [2].

It is the most stable system for portable devices. This means that it is resistant for crashes. Palm OS can be run on the very weak hardware that has small memory and processors resources. A good example is Zire 31. It has only 16MB build in memory and 14MB is available for user's applications and data. It weight is only 136g [14].

Figure 3 shows how a typical Palm OS screen looks like.

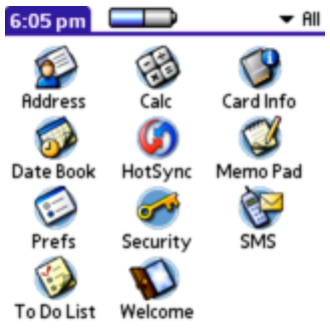


Figure 3. Typical Palm OS display [14]

3.2.3.2 Linux for mobile devices

There are only a few devices on the market that are running under Linux control, for example: iPaq and Zaurus. They have similar an interface to the desktop version of a Linux system, and based on it, so it has all advantages of Linux Operating System. As a result it allows running most of typical Linux programs and in most of cases they should be only compiled using a special compiler that can generate output code understandable by Pocket PC or Smartphone processor. The next advantage of using Linux is the fact that it is open source so there is no extra cost for licenses for the operating system.

It has the same advantages as the desktop version and also the same problems (drivers for different devices are not supported by manufacturers and most of them is written by enthusiasts – this is the reason why at the device that has Windows Mobile installed, the user can not just change operating system to Linux) [2].

There is a few Linux distributions that are prepared to run on Smartphones or Pocket PCs. Available commercial distribution is Qtopia [15] and open source distribution is known as OPIE [11].

Minimal system requirements (based on Qtopia requirements) are:

- for Smartphones: ARM processor (and others [15]), 32MB Flash ROM, 32MB RAM
- for PDA: ARM processor, 8MB Flash ROM, 16MB RAM

Figure 4 shows a typical screen shot from Linux for mobile devices.

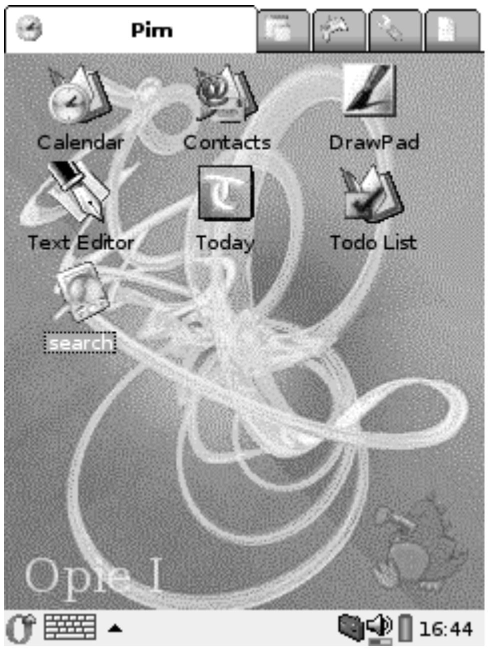


Figure 4. iPaq 3100 powered by Linux [13]

3.2.4 Choosing software platforms

Each of the presented operating system has different advantages and disadvantages. Symbian OS and Palm OS are from the beginning designed for mobile devices and phones and they can only run on those devices. They have small hardware requirements and like Symbian they are the most popular. One disadvantage is that they need to create software from beginning and using SDK that strongly depend on those systems and their versions (Symbian provide some independent on entire platform but moving software from UIQ to Series 60 can be sometimes very hard). Windows has other features. It is a big company that stands behind that system, and they have much experience in operating system design. The interface is very similar to desktop machines and developers can use .NET framework that is almost the same like for desktops (developers do not need to learn a new technology). A main disadvantage of this system for me is that there is no close button in most of applications (typical x button in corner just minimize application, not closing it). On the other hand there is a Mobile Linux that based on GNU license so it is for "free" (manufacturer must pay only for distribution and their special features). In theory this system should allow run most of typical desktop applications without any changes, they need only to be compiled for the ARM (Advanced RISC Machine) architecture. This kind of processor architecture is power safe and those kinds of processors are used when low power consumption is a critical design goal (like in mobile devices).

When I thought for which system design my master thesis project I closer look what devices are having my friends and people that I can met. Symbian OS is the most popular operating system for phones and it is supported by Nokia, Ericsson, Motorola and other phone manufacturers, Most of people have phones with this operating system but they do not have built-in GPS receiver (only two Motorola phones have it).

I choose Windows Mobile because it is also the most popular operating system on palm market and in most of those devices GPS receiver is included. Because of that it can be used as navigation devices. Other operating systems for palms are rather very exotic and now only one manufacturer build devices with this system.

Two software platforms were selected but only Windows Mobile version of application was completely designed. It was easier and faster to design application for this platform because I have device with this system. My Symbian OS device is rather old and has only 1MB free memory so it is very hard to fit with map and application in this size. Graphic and performance tests were performed for Symbian OS devices and results from those test allow to be sure that device with more memory will be enough to run the designed application. This test concentrate on drawing abilities of Symbian OS devices. They allow to draw ten thousand of random figures, filled random patterns in fifteen second (Nokia 7650 phone was tested).

4 MAP – HOW TO ORGANIZE DATA STRUCTURES

Gathering and storing data to draw a map is not an easy process. There are raster and a vector maps available. The first ones are not interesting in case they need to be used in the path finding process. To determine the path and calculate its length vector maps are needed.

The first problem is to get specified map of a region or a city. It is possible to buy such map. It can be expensive and probably do not solve the problem because each map has different formats. To use them the format must be handled by the application or a map must be converted to the supported format. Another problem is that maps will not have Points of Interests so they needed to be added manually. Because of that for this thesis I decided to make my own maps of two cities: Wrocław (Poland) and Karlskrona (Sweden). It allows me to know how to build a map and what the biggest problems are. The other argument for this is that my application needs a lot of interaction with data. It is not like in existing path's finding software. It needs to operate on the data looking for objects that can be seen while building a path. I created simple software to build maps. It allows adding points, streets and areas (it will be described in section 0 of this chapter).

The second problem is how to organize data that will be used to draw the map. It needs to be displayed on the screen and allows changing zoom and rotating. The application should also allow scrolling the map (it is impossible to display the entire map with details on a small screen).

First requirement is that data structures used to store the map should allow fast choosing of a visible part of the map (segments). The second requirement is that path finding algorithms need a stored graph of groups of points. Because the map does not consist from streets only, I also decided to have availability to display areas like:

- water (seas, rivers, lakes)
- parks
- islands (lands, islands on sea, islands on rivers and lakes)
- buildings
- districts (not visible object)

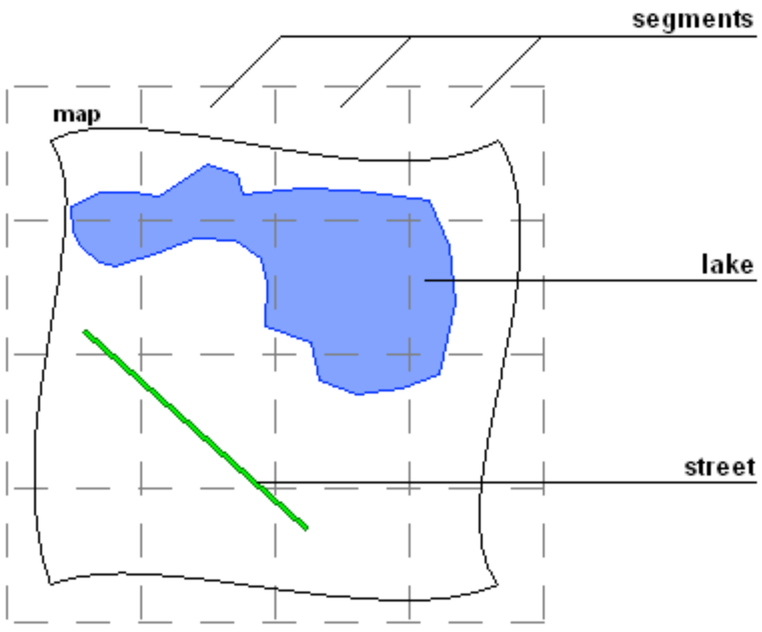


Figure 5. Example map with lake and street

I chose to split a map into segments. Each segment will represent a rectangular part of the map and store information about objects on it. Streets on the map will be represented as points and connections between them. Areas like islands, lakes and other will be represented as table of vertices. Objects will be assigned to segment if they should be displayed in this part of map. From this point of view one street or area can be assigned to more than one segment and it should be done without storing all object data in segment. Figure 5 illustrates the problem. On the figure one lake and one street are presented.

To solve the problem with redundancy I made a collections for points, streets and areas and in segment only pointer for specify object is stored. Having a pointer to an object allows to get its data. Figure 6 presents the data model for map representation.

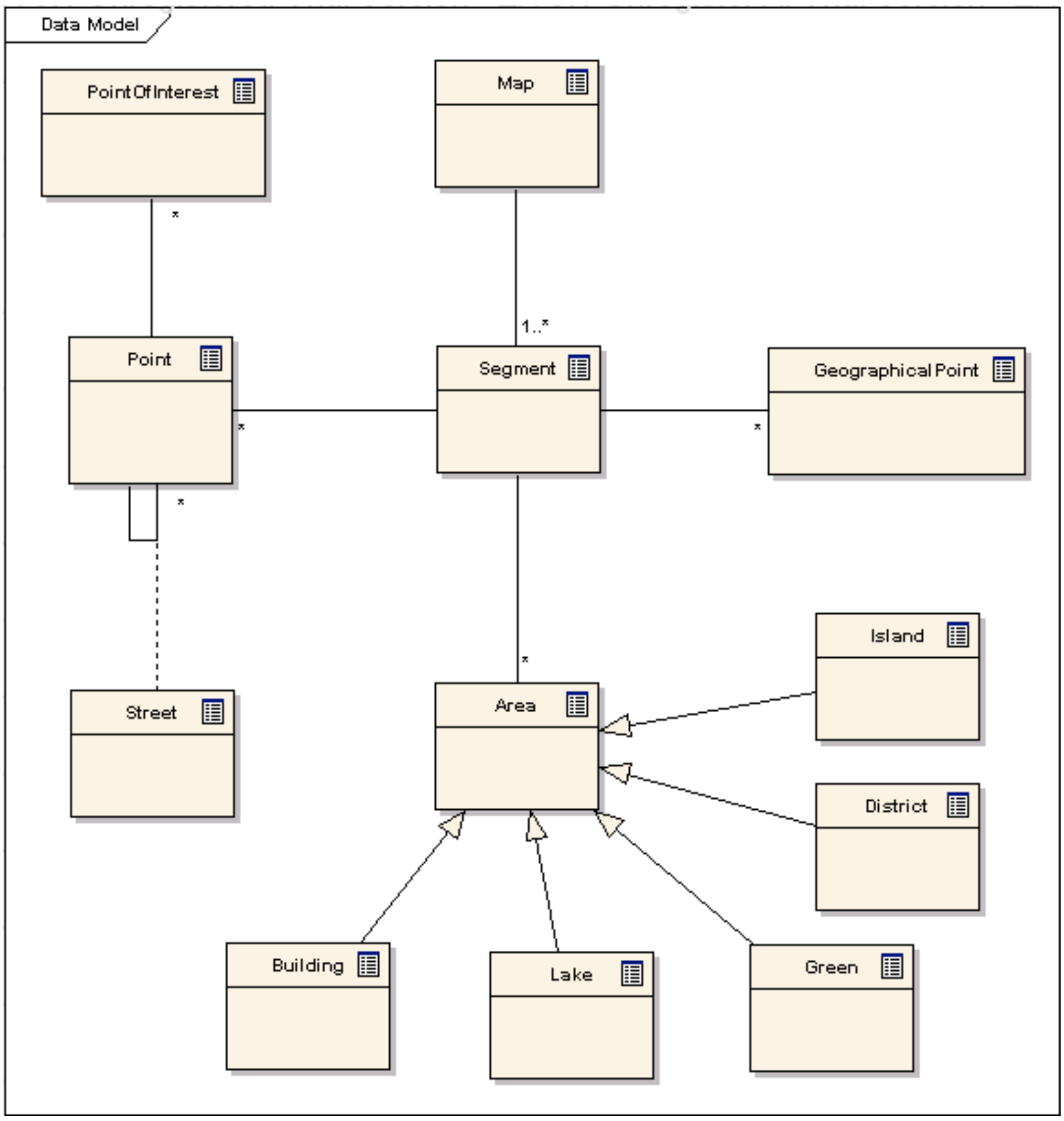


Figure 6. Database diagram for map data

Because a map needs to fit in phone and palm memory each point has integer coordinates. This type of data uses 4 bytes per value and the fact is that calculations on integer type are very fast (zoom in, zoom out, scrolling – all those operations need few mathematical calculations). Only few points with geographical coordinates like longitude, attitude and height above the sea level will be stored for a region (these are float values, longer and using them in mathematical operations consumes more time). Coordinates are interpolated for every point in a segment. I can make this assumption without loosing accuracy because segments will be relatively small and because of that interpolation of geographical coordinates is a linear process (spherical shape of the earth can be ignored for small areas).

4.1 Streets

Streets will be represented as map lines. This means that one street can be created from at least one map line. In typical case it will be few (it depends how many corners the specified street has). One map line is a connection between two points. This connection will have extra data that describe it, like length, street name, type of connection (road for bikes, one way road, only for pedestrians). Also other additional information like maximum speed can be added if needed.

To determine which streets should be displayed on the screen, segments' visibility will be checked first. An example is presented on Figure 7. This figure presents my first approach, when only information about points was stored in a segment. In this case only black points will be checked if they should be displayed or not and lines that connect those points with others will be drawn.

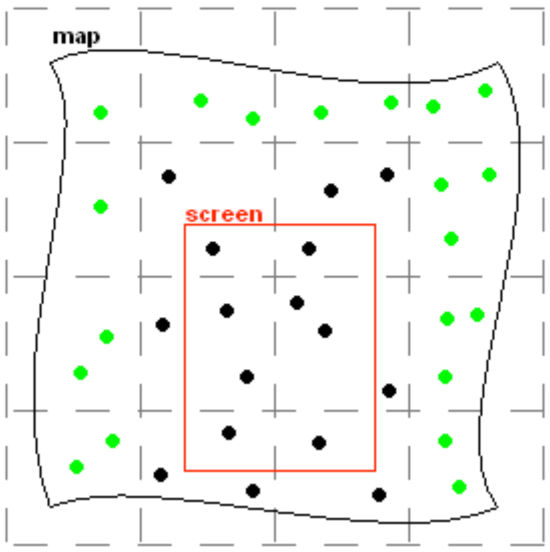


Figure 7. Point's visibility

Map line will be visible when one of points is visible. This situation is presented on Figure 8.

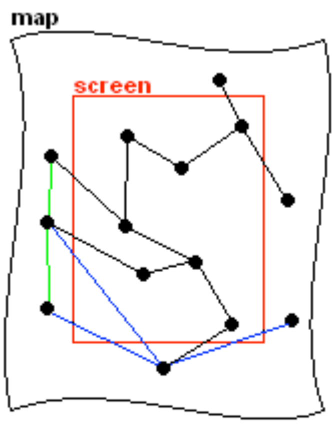


Figure 8. Example map with streets

Black lines will be visible on screen; green lines are out of screen so they will not be visible. Blue lines should be visible but in this case they were not, because none of the points which represent those lines is on screen. When I found this drawback, I decided to add information about streets to the segment. Actual displaying algorithm checks all streets that are in the segment, not only those which have one of their vertices inside it. Figure 9 presents how a map is displayed after adding information about streets in a segment.

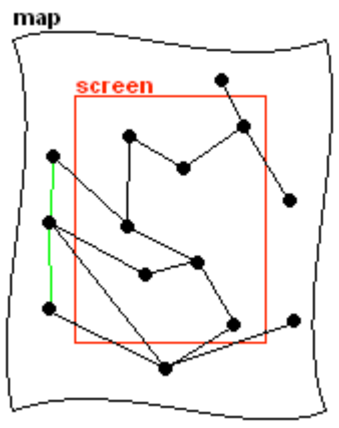


Figure 9. Corrected map

This solution increases amount of data needed to display the map. Each segment must store pointers of map lines that should be displayed in it. As I show, it is really needed to display the map properly. The fact is that a pointer is only an integer value and map lines will be rather short so redundancy will be minimal (there are not so many lines that are in more than one segment).

4.2 Areas

An area is an object which should be displayed on map. It can be one of following region:

- seas, rivers, lakes
- islands (two types, islands on seas and islands on rivers and lakes) or lands
- parks and other green places
- buildings
- districts (not visible on map)

All of those regions do not exist in every city. Developed application was not built for specific city but it is able to work with different maps and should accept all possibilities. It is able to handle all maps for different cities.

As it was explained at the beginning of this chapter, an entire map is divided into segments. Some example situation is presented on Figure 10.

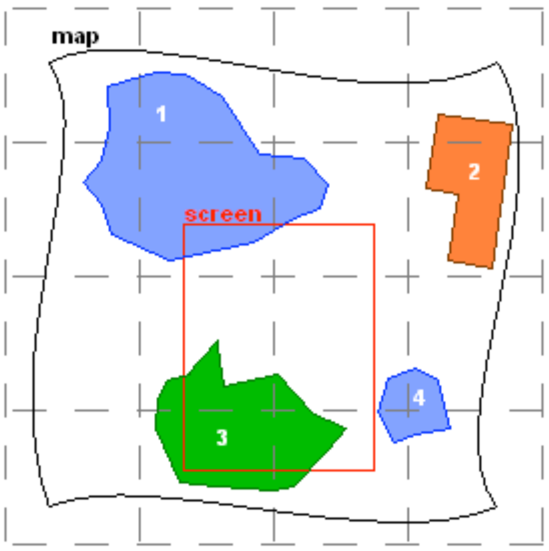


Figure 10. Areas on map

In the illustration above areas 1 and 3 should be displayed on the screen. Before drawing map the segments' visibility will be checked. This produces a set of segments which will be visible. Then for each of those segments areas that are in it are checked if they should be displayed. In this case areas 1, 3, 4 will be checked and only 1 and 3 are drawn on screen. This organization of data guarantees that only few areas will be checked and the processor time will not be wasted to check all of them if they should be draw or not. Figure 11 illustrates this situation.

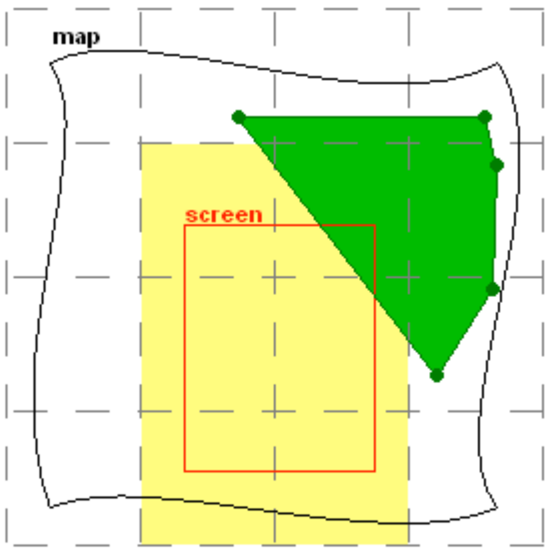


Figure 11. Area example

Assigning an area to the segment is a bit difficult in map building process. To determine if a polygon is inside a segment or has some part inside it few calculations must be done. Those calculations are divided into three steps:

1. adding each polygon to all segments in which its vertices are
2. adding segments that are not added in step 1 but at least one of the polygon's edges has a common point with segment edge
3. adding those segments that are entirely inside polygon by checking if a centre of the segment is inside the polygon (a centre or other point from a segment can be checked, it

is not important if segment edge does not have any common point with polygon edge and if no vertices are inside segment)

Another problem with areas is that sequence of drawing them is very important. It is connected with areas' drawing order. For instance if an island will be drawn at first and after that the sea surface, then the island will be invisible. Because of that sequence of drawing is very important. The order is following:

1. areas:
    a. seas
    b. islands
    c. lakes and rivers
    d. islands (on a lake or a river)
    e. buildings and green places
2. streets
3. Points of Interest

As it can be seen there are two kinds of islands. It is because it is possible to have big island on sea with lake on them and a small island on the lake. All of those objects must be visible and to handle those situations, a proper sequence of drawing is needed. This order means that drawing procedure needs to enumerate all visible segments five times looking for areas. At each stage it selects different objects to draw (drawing layer by layer from bottom to up).

4.3 Point of Interest

Displaying Point of Interest (POI) is very easy. It is a single point on the map and it is represented as a filled circle. From this point of view, they are the easiest object to determine their visibility on screen. POIs have a special availability that allow click on them and show additional information about.

Data structure collect all needed facts about them and also allow to search in the POIs set using different criteria. POIs are grouped by categories. In the current version they are:

- museums
- art galleries
- sport activities
- parks and green places

Each POI has two very important attributes:

- the priority – it is value which describes relevance of POI from interesting point of view. Objects like famous museums, national wonders, castles and other have this attribute set to highest (must be seen). Te other values for it are (in sequence from very important to less interesting): interesting, normal, low, very low, no priority. This parameter is used when POIs that should be visited are searched. It depends on the person which is responsible for preparing the map so it should be set very carefully.

- the visit time – this parameter describe how many time is needed to visit POI. Each POI has three times, which represents difference style of visiting. They are: fast, normal and slow. The user elect which time should be used when he choose the trip type (see much as possible, few objects or through visiting).

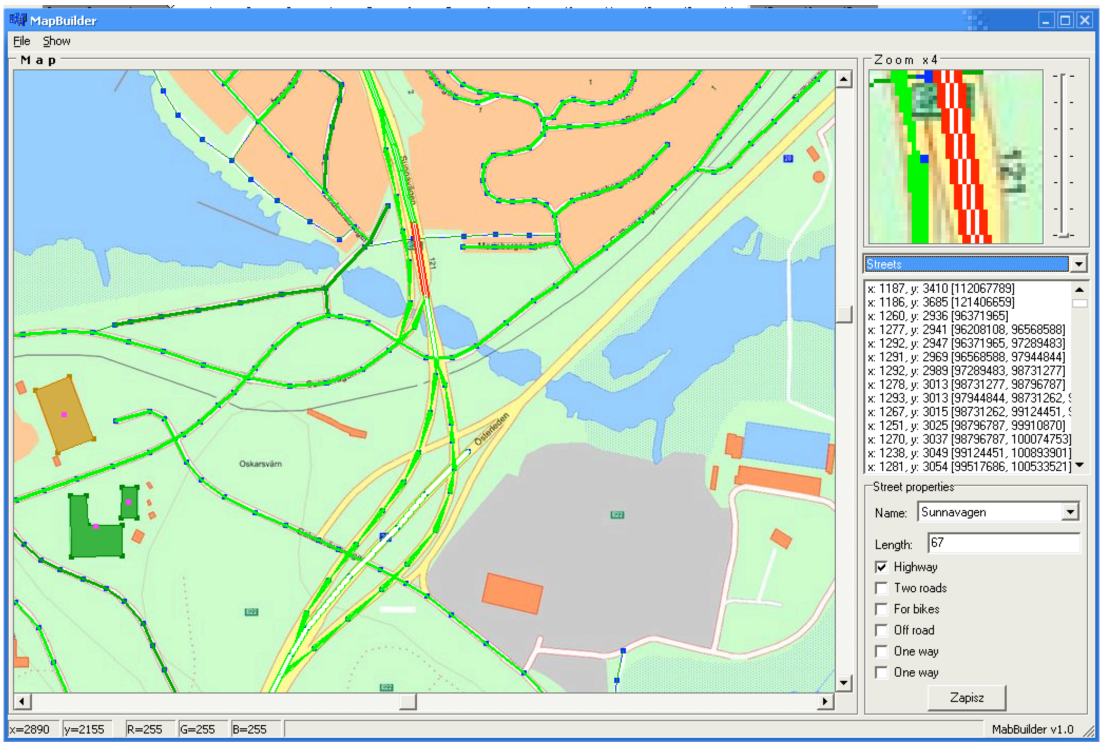
4.4 Main problems in the map creation process

Making a map from scratch is a complicated process. Gathering all the data that allow drawing a map with limited accuracy is very difficult and consumes a lot of time. It should be automated or some existing maps should be used but I want to know how to do this and how it can look like. From that reason I chose doing it in way that is building a map based on some raster map and then construct a vector map.

Another problem is how to organize data structures. This strongly determines the final size of memory needed to store the map. It also determines how fast algorithms that are operating on the map can be. The chosen solution increases the amount of memory but should allow faster drawing of maps on the screen. It is very difficult to accept for the end user when he needs to wait for simple operations that probably he will do very often (for example scrolling map). Today phones and other mobile devices have slots for memory cards so efficient data processing is more important that minimizing size of data.

4.5 MapBuilder software

This software was written by me using Borland C++ Builder 6.0. It allows making a simple map. In this process I took the raster map (image) from the webpage *http://eniro.se* (it is used only for this thesis so it can be used for free). I made one big bitmap with the map, read it with my software and then I added streets, areas manually. It took a long time and demanded precision. This process should be automated but it is not a part of this thesis. Work which has been done in this thesis can be a beginning of making this process automated. Figure 12 presents main screen of this application.



```
x: 1187, y: 3410 [112067789]
x: 1186, y: 3685 [121406659]
x: 1260, y: 2936 [96371965]
x: 1277, y: 2941 [96208108, 96568588]
x: 1292, y: 2947 [96371965, 97289483]
x: 1291, y: 2969 [96568588, 97944844]
x: 1292, y: 2989 [97289483, 98731277]
x: 1278, y: 3013 [98731277, 98796787]
x: 1293, y: 3013 [97944844, 98731262, 
x: 1267, y: 3015 [98731262, 99124451, 
x: 1251, y: 3025 [98796787, 99910870]
x: 1270, y: 3037 [98796787, 100074753]
x: 1238, y: 3049 [99124451, 100893901]
x: 1281, y: 3054 [99517686, 100533521]
```

Street properties

Name: Sunnavagen

Length: 67

☑ Highway
☐ Two roads
☐ For bikes
☐ Off road
☐ One way
☐ One way

Zapisz

x=2890 y=2155 R=255 G=255 B=255

Figure 12. MapBuilder application with a part of Karlskrona map

Green lines indicate streets. Figure 13 presents how street with specified attributes looks like. This software allows adding the following attributes for roads:

a) highway
b) express road

c) typical road
d) one way road
e) road for bikes, pedestrians, also no car traffic attribute can be assigned
f) like typical road, but this colour means that the street do not have specified name (to fast determine unnamed streets)

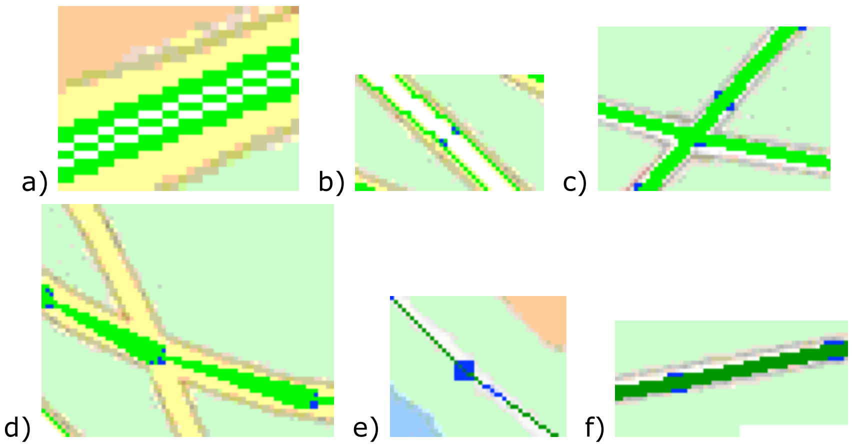


Figure 13. Roads example

As it was mentioned before, the software also allows creation of visible areas like:

- seas
- islands
- buildings
- districts
- green places like parks
- rivers

Some of them are presented on Figure 14.

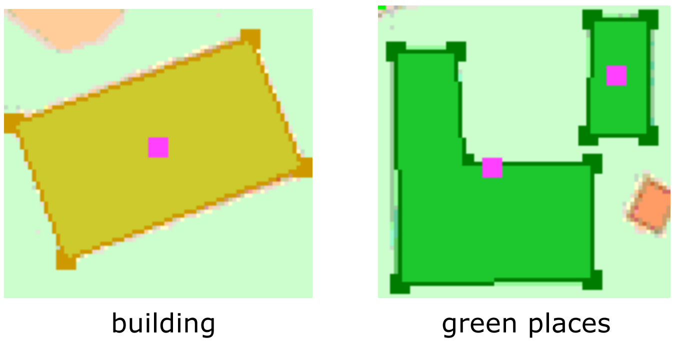


Figure 14. Areas example

The software allows working in one of following mode:

1. streets
2. areas
3. POIs

In each mode user is able to work with different kind of objects but he always works on points. Objects can consist from one point (POIs), two (streets), three or more (areas). Actions like adding new point, moving it and deleting are available. This allows to build a simple map. For each objects user can choose depending on object's category:

- for streets: name, street's attributes (look at previous page), length (calculated automatically)
- for areas: name

- for POIs: name, POI's category, description, photo, time for visit (three values are stored: fast, normal and short visit time)

This allows building the fully functional map for the designed application.

4.6 Displaying map in mobile device

Figure 15 presents the map that is displayed on the palmtop device. It has different colours from the software used to build map. The main reason is that screens in portable devices are limited than desktop computers have. A map should be visible when it is dark and also when the sun is shinning. By choosing colours for mobile devices I try to make the map visible in all possible situations.

Users can interact with a map by scrolling it using drag & drop feature or scroll buttons. There is also functionality to zoom the map in and out, so it is possible to see almost the entire city on the screen (it is possible for cities that are similar in size to Karlskrona).

Drawing a map takes some time so a progress bar is displayed. Map buffering is also used. The application uses memory bitmap to draw a bigger map than it is possible to fit on a device screen. When users scroll a map, only a part of the memory bitmap is copied onto screen. If users moves the map and a part that is not available should be displayed, then the application draws a new piece of the map in the memory (whole memory bitmap is refreshed). As it was said at the beginning of this chapter, a map is divided for segments so refreshing the map requires to check which segments should be displayed and then to display streets and areas that are inside those segments. While drawing a map, a Z-order must be considered. Establishing which object should be displayed and which will be obscured by the other is meant by that, for instance islands on a sea or lakes, buildings, streets on the land. To manage this, the application starts drawing from bottom layers to top ones. The order from background to first plan is:

1. areas:
   a. sea
   b. sea islands
   c. rivers, lakes
   d. islands on river or lake
   e. green areas, buildings
2. streets
3. POIs

While enumerating which segment should be displayed, the drawing algorithm takes every object from a segment and puts it in the list that corresponds to object's layer. This list is sorted and each object in it has a unique key. This key prevents drawing the same object twice or more (for instance if a street starts in one segment and finishes in the other, it will be only added and drawn once – structure used to store objects do not allow to duplicate keys).

Drawing a map depends on the zoom level. If users zoom out, then more objects are taken into consideration. This is the reason why drawing the map with maximum zoom-in is faster than in any other zoom level. On a palm device drawing a map takes not more than 2 seconds when maximum zooming out is set.

To prevent users from refreshing and drawing map after each scroll operation occurs, memory bitmap is used. Problem with refreshing the map can be seen in the TomTom or other application. They draw map and generate delays while scrolling. My application refreshes map immediately after a scroll. The map is buffered in memory and just copied onto the screen. When users cross boundaries of the map buffered in the memory, then a drawing event occurs and a new map is drawn in memory. This technique needs some extra memory but as I decided, the application speed is more important than the memory consumption.

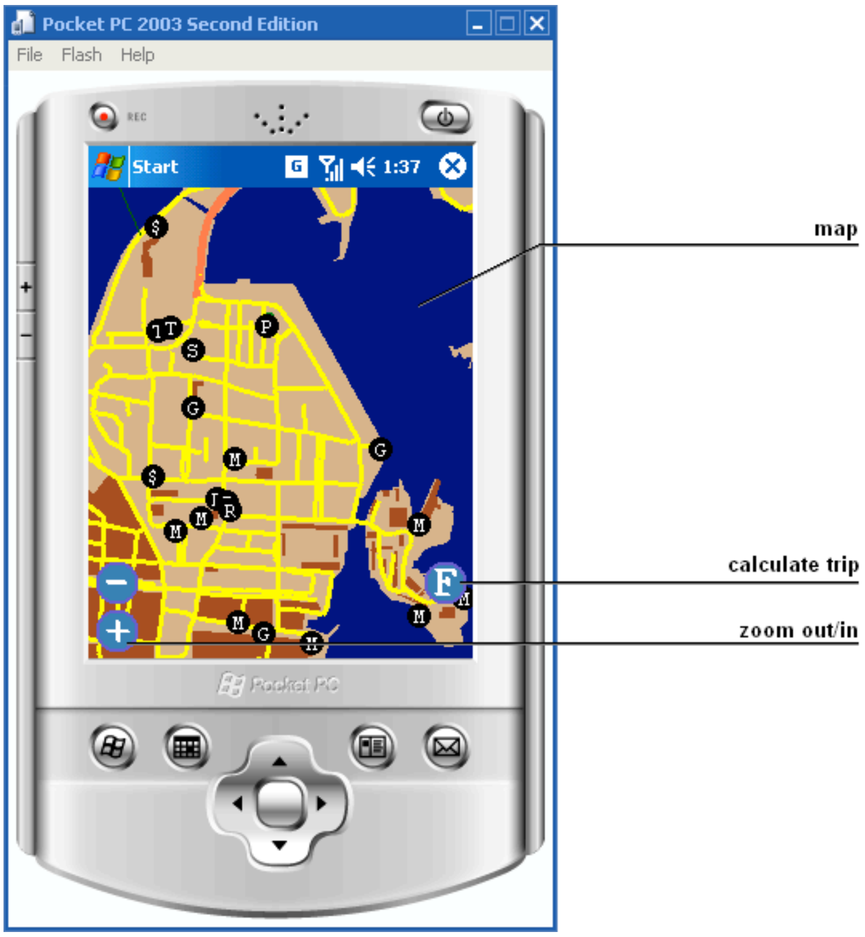


Figure 15. Intelligent Tourist Information System

5 FINDING PATH

From the mathematical point of view the map is organized as a graph. It is represented as a set of vertices. Each vertex has a neighbour list which stores connections to all vertices reached from this one. Information about edges is stored in streets set. This set is organized as a sorted list where keys are line coordinates (two points' coordinates between which this edge is). This organization of data prevents from duplication about each edge and storing it in two places. For instance if there are two points A and B and they are connected information about the edge is stored in other structure and it do not need to be stored in point A and in point B.

From this point of view problem of finding path looks like a typical graph problem. There are a lot of solutions for that and they can be found in [1] and [3]. There are presented algorithms to find shortest paths from one point to others. In the system that I designed the main problem is that it is difficult to determine which points should be in path. Objects that tourists want to see can have different priorities and different time needed to visit them. They can be on a very small area or they can be scattered throughout big territory. The problem is to find an interesting set of points and some short path between them. Those points can not be determined before finding path process starts. They must be dynamically selected while building path process is performed. Because of that to calculate the trip's path I decided to use two algorithms that can be found in those books. The solution is a combination of path's finding algorithms and observations that I have made while travelling. I decided to split the problem for smaller tasks and then solve each of them well knowing techniques. This idea consists of four steps:

1. Selecting possible targets
2. Fast path finding
3. Smoothing path
4. Finding path on map

In next paragraphs I describe more about all of those steps. I will also mention about problems and how my application can manage them. The presented solution produces some paths that are acceptable and can be used by users as a base in a planning path process. This means that they consist of selected points according to user's criteria, and trip time fit in a period specified by the user. The order of the points in the trip might be probably changed to make the path shorter[1] in step 3.

There is one assumption that is needed to be sure that there is always a possibility to find a path. The graph that represents the map is consist. This means that from every point on map is some path to all other points [3].

5.1 Selecting possible targets

The main idea of the system is to give a good starting path for visiting a city with minimal user knowledge about the city and about available attractions and activities in the city. I wanted to build a solution that give people ability to come to a new place, a new city and spend a nice time there.

[1] It is not optimal because this depends on user. For one person results can be satisfying but for other they can be bad. The idea is to build few propositions for trips, show points included in it and allow user to know something more about them. This should give knowledge for user and allow him build another path basing on results produced by the application. The application builds some "good" solutions not random paths.

From this point of view the user should be able to select trip criteria. He can determine how much time he has, what kind of transportation will be used and his personal preferences for visiting objects. In the existing version there is a time constraint for visiting objects. This constraint is connected with time needed to visit each objects and style of visiting. User has possibility to choose from three options:

- fast sightseeing (much as possible objects will be seen)
- typical sightseeing (typical visiting time)
- precisely sightseeing (a long time for each object, that allow to know a lot of about each place)

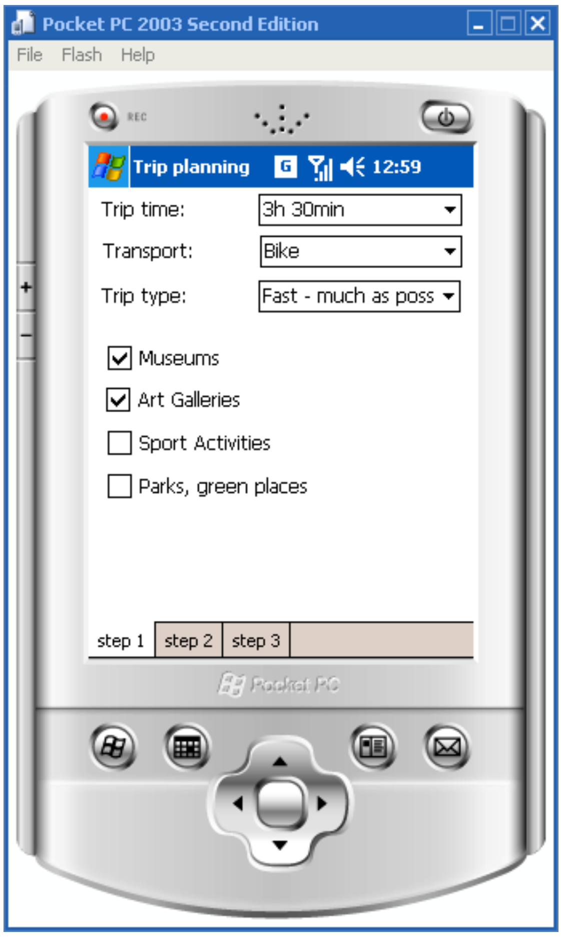
Each of those is based on values that are stored in every POIs and determine three times for each object. A time criterion for visiting objects is very important because the *building path algorithm* made by me must fit in time specified by user.

I mentioned that transport type (walking, biking, and driving) is also specified by the user. In the existing version of the program it is used to calculate time that is needed to travel between two objects. This parameter also determines which roads can be used to travel from point to point. It is very important if tourists travel by car (in future versions maybe by a bus). It must be taken into consideration that in most cities there are one way roads and areas closed for vehicles. See 6.4 for more information about this parameter and how it can be used in future versions.

The user must also select criteria that describe his preferences to the objects that should be visited. On *selecting possible targets screen* the user can select only categories of objects. There are four groups of POIs available:

- museums
- art galleries
- sport activities
- parks (botanic gardens, zoo and so on)

On Figure 16 the *planning trip criteria window* is presented.


Pocket PC 2003 Second Edition

File Flash Help

Trip planning 12:59

Trip time: 3h 30min

Transport: Bike

Trip type: Fast - much as poss

[✓] Museums

[✓] Art Galleries

[ ] Sport Activities

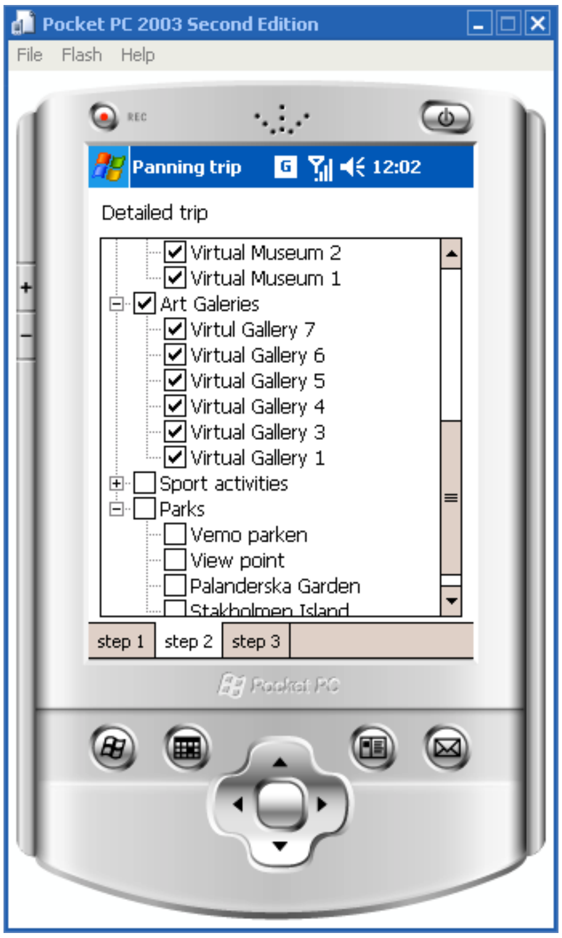
[ ] Parks, green places

step 1 | step 2 | step 3

Figure 16. Determining the trip criteria

It is a first step to find the trip. In next step the user has the ability to select or deselect each of objects. This give the possibility to exclude some points that user does not want to see or had seen before. Selecting object in this stage does not guarantee that this object will be visited. This only means that set of selected objects will be used as input for the algorithm to find trip. Figure 17 presents the screen that allows performing those activities.

Objects are grouped in categories and they are composed as tree where root is a category name. Functionalities like selecting category or deselecting it by root node is available. At this stage a category which was previously not been chosen can be selected here. The same functionality is available for each POIs.



Detailed trip

- ☑ Virtual Museum 2
- ☑ Virtual Museum 1
- ☑ Art Galeries
  - ☑ Virtul Gallery 7
  - ☑ Virtual Gallery 6
  - ☑ Virtual Gallery 5
  - ☑ Virtual Gallery 4
  - ☑ Virtual Gallery 3
  - ☑ Virtual Gallery 1
- ☐ Sport activities
- ☐ Parks
  - ☐ Vemo parken
  - ☐ View point
  - ☐ Palanderska Garden
  - ☐ Stakholmen Island

Figure 17. Detailed choosing objects

5.2 Fast path finding

The path finding algorithm is based on my own experiences. When I want to visit a city I take few brochures from tourist information centre, ask friends and search the internet. This knowledge allows me to choose number of objects that I want to see in a city. I construct a “good” path between those points and then if there will be enough time I try to find some addition objects that can be visited while moving from one object to another. The implemented algorithm is performing the same operations but the path produced by it is rather “good enough” than optimal (as it was mentioned it should be only a proposition for trip).

To make the calculations faster I make an assumption. Distance between POIs is estimated as distance on a straight line multiply by √2. The motivation for this is that in most cities this will be similar to real path because streets are perpendicular and parallel (or almost perpendicular and almost parallel) and a connection between two points in a city in most of cases can be presented as two perpendicular lines (more or less). It gives enough accuracy and is very fast.

The main idea of the implemented solution is presented on Figure 18 and Figure 19 as blocks. As can be seen from the figures the algorithm is basing on dividing problem for smallest. It is recurrence algorithm. At first two POIs that have the highest priority are selected. This process is based on determining the highest priority for POI in POIs database and then selecting the point that has this priority. The second point is selected in the same way (the first selected point is excluded from POIs database). After that, time needed to visit the selected POI is calculated. If this time is smaller than time entered by the user the algorithm run the recurrence part (Figure 18). First found POI initially “visited list” (let be A), the second is added to “POIs to visit list” (point B). The path lengths between those points call it x variable. After that the *find between* function tries to find some POI between those two. In this case a set of point is selected using distance criteria. A set of POIs that path length from A to each of them and to B is not bigger than 1.5 * x. From this set POI with highest priority is selected – let it be a C point. After that time is calculated (for travel and for visiting) and if it stills enough the function run itself to find something between A and C. This action is performed also for C and B. It is needed to find other POIs on the second part of trip (another point between C and B). Process of building trip is presented on Figure 20.

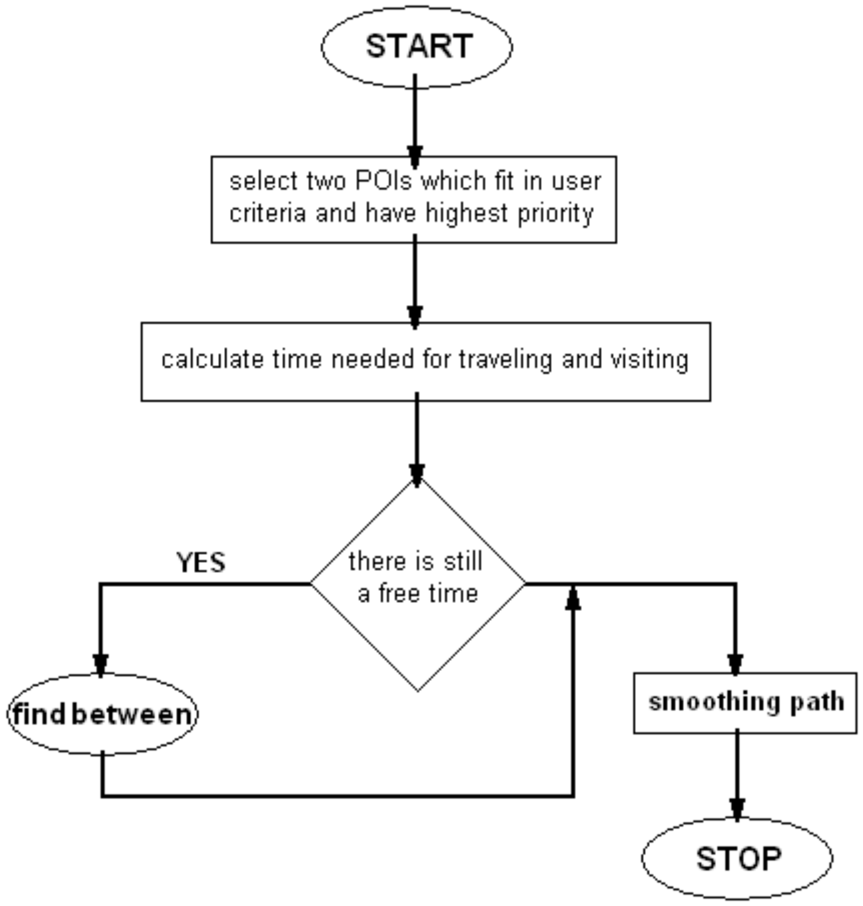

START
select two POIs which fit in user criteria and have highest priority
calculate time needed for traveling and visiting
there is still a free time
YES
find between
smoothing path
STOP

Figure 18. Finding path algorithm

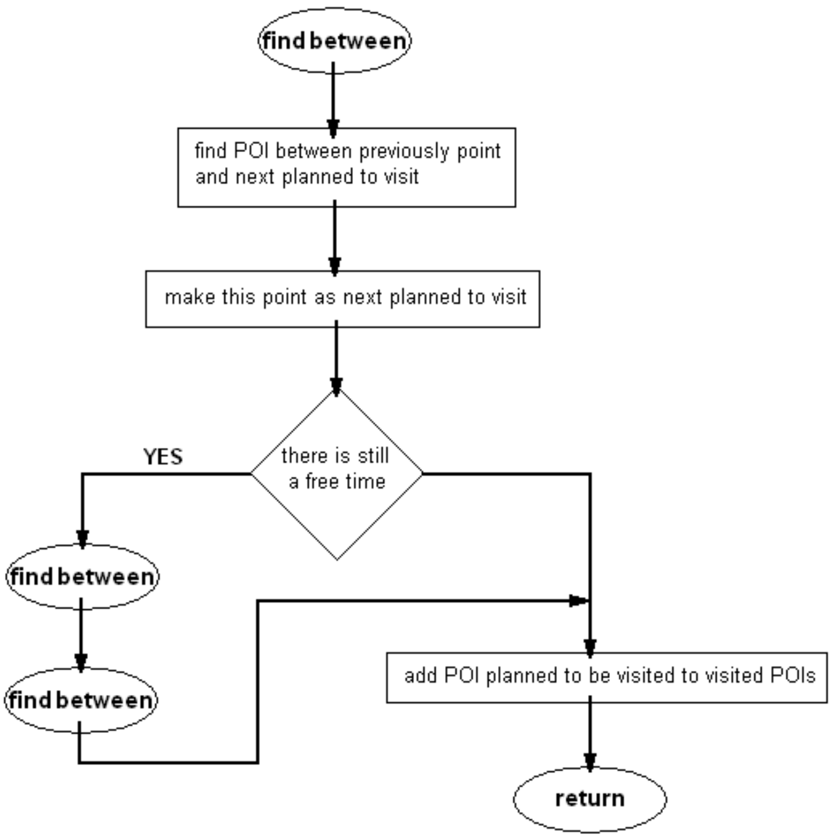

find between
find POI between previously point and next planned to visit
make this point as next planned to visit
there is still a free time
YES
find between
find between
add POI planned to be visited to visited POIs
return

Figure 19. Algorithm for "find between" function

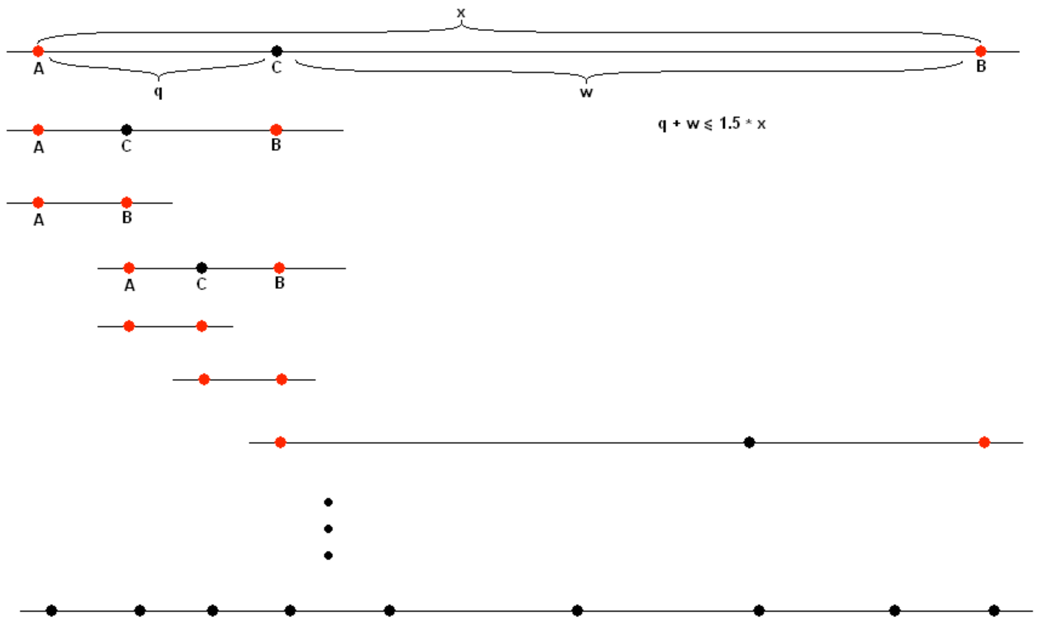


Figure 20. Building path for trip

The red colour marks two points between which a third point is looked for. The implemented algorithm searches for POIs using few constraints. First and most important is that the road to the found POI and to next can not be longer than 1.5 of original road between two POIs. The value 1.5 was estimated after few experiments. Smaller values reduce the number of possible POIs to visit (1 means that third point – C must be exactly on the current path between A and B). Bigger values produce sometimes strange and confusing results.

The next constraint (presented on Figure 19) is that the total time of visiting and travelling can not be longer than specified by user.

The constructed path for trip is not optimal (as it was build according to presented above steps) and it needs some "smoothing" operations. On Figure 18 block which perform this operation is located before STOP. It is described in the next paragraph.

The presented algorithm selects two POIs with the highest priorities. After that the algorithm returns the path and finds another path for other two POIs. All combinations of two POIs with highest priority are performed. This is because path strongly depend on place where it start and where it finish. From this point of view making all possible combination of start and end POI is a good way to find better trips.

The entire calculation does not take more than one second. The criteria specified for this test were:

- trip time: 10h
- transport: vehicle
- trip type: fast

Those criteria produce the biggest and the longest paths with a lot of POIs – something about 20 for each path. From the user interface perspective (time aspect) it is good and fast enough. This stage also produces a lot of paths (something about 100 – all combinations with the highest priority POIs as beginning and end).

5.3 Smoothing path

The smoothing path algorithm gets a lot of paths from the previous step. Most of them have strange connections and some of them are less interesting than the others. The smoothing path process takes all generated paths and counts the average priority value per POI for each path. It is counted summarizing priorities for all POIs in path and then dividing it by number of POIs in path. In next step the algorithm selects the most interesting path (it is this one which has the highest average priority). Also the longest path is selected to determine the maximum visited number of POIs per trip. Then all paths that have average priority less than 90% of typed maximum are eliminated (removing less interesting). After that trips that consist from less than 80% POIs than the longest path are removed (removing short paths). This brings from 5 to at best 40 paths (depends on criteria – for some can be a lot of similar paths). Only the first 10 paths are displayed (sorted by average priority). This constraint is used to prevent user from wasting time on browsing and comparing trips. In fact if the user does not find a satisfying result among the first ten paths mostly he is changing the searching criteria. This truth is taken from the web searching engines. There is a lot studies about that.

Selected trips are not shortest paths and they need to be smoothed. An example is presented on Figure 21.Only the most interesting part of the trip is shown. Black filled circles with letters represent POIs, red show POIs that will be visited. Number near POI is the order of each POIs in trip path. As it can be seen on the Figure 21 the path is not optimal. Changing order of few points can make it shorter. This idea is used to make paths "smoother".

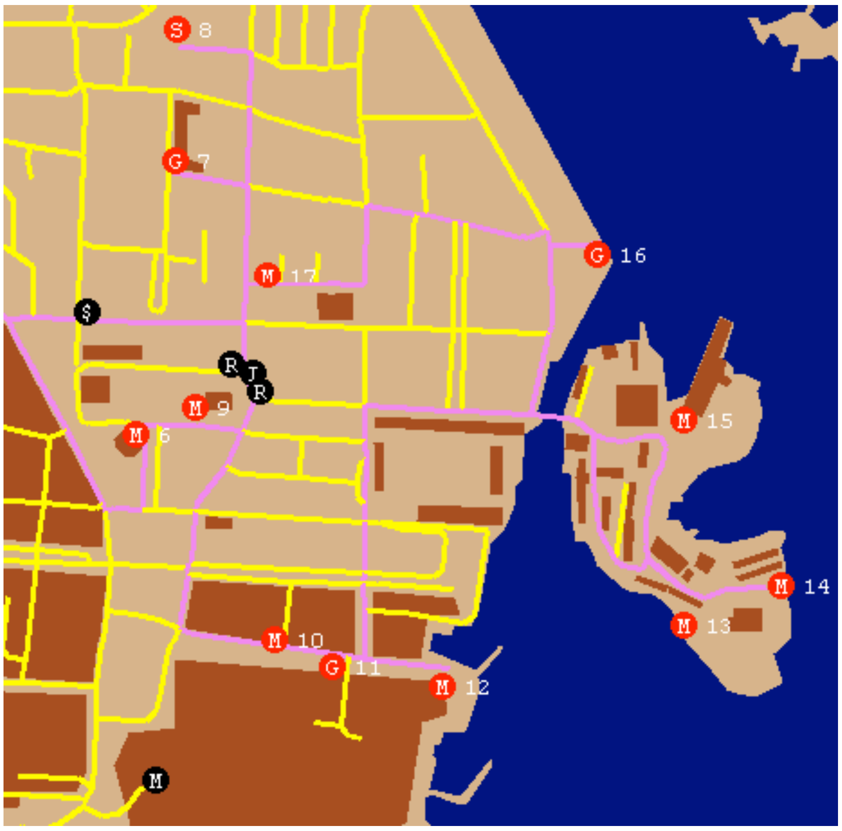


Figure 21. Example trip without optimization

This idea is very similar to the idea of bubble sort. Because the path is almost as short as possible it needs only small corrections to the algorithm and it will be good enough[2]. The obvious thing is that paths that consist of one or two points are not taken into consideration (they are shortest as possible). The basic idea is very simple. There are four steps:

1. First the algorithm checks the first three points, for instance A, B, C. The actual path is A →B→C. If path B→A→C is shorter then the order of the two first points is changed.
2. For each four points in sequence path A→B→C→D is compared to A→C→B→D. If the second is shorter then points B and C are shifted.
3. This step is similar to the first step but move last three points. If path A→B→C is longer than the path A→C →B then B is shifted with C.
4. If order of points was changed algorithm goes to step 1.

The method is very easy and of course this does not produce the shortest path but can make it shorter than it was previously. It is not shortest path because it operates only on two points which are neighbours. Figure 21 presents this situation. When we look a this figure the better path for this is 8, 7, 17, 9, 6, 10, ... The path was not changed because the method does not take into consideration and does not compare paths between more than two points. In this case paths between 6, 7, 8, 9 and 17 should be considered and checking all combinations of paths between those points allows finding the shortest path. It can be improved by adding some extensions and made the algorithm that will be able operates on more than two points.

The typical cycles needed to optimize path are from one (no changes) to six – the longest trip variant with fastest visiting time (experimentally checked). The average value

[2] Analogism to bubble sort is used because it is very similar and also bubble sort can be a fast solution when objects are partially sorted [3]

was not approximated but it should not be more than four cycles per trip (for 23 trips the average cycles count was 3.7).

The complete solution for this problem is presented in [3]. Floyd-Warshall's algorithm can be used to find the shortest path between all points. I considered it implementation but it has cubic complexity [3]. This mean that time needed to solve the problem grows with the cube of the number of considered points. Because of that I decided to do not use this algorithm.

5.4 Finding path on map

Calculating real path (the path on map, described by streets) consumes a lot of processor and memory resources. It is the process when real path between two points on map is calculated. To solve this problem I use Dijkstra's algorithm presented in [1], [3], and [8]. It was originally developed by Edsger Dijkstra and is one of greedy algorithms. It can be used to solve single path problem from single source on directed graphs with non negative edge weights.

Dijstkra's algorithm use the observation that shortest path from A to B consists of shortest paths to each of points that are on it. For instance if shortest path from A to B run sequentially on vertices C, D, E,… this mean that A → C is shortest path to C from A, A → C → D, is shortest path from A to D and so on[3].

This algorithm operates on set S and array D. Set S include the visited vertices. Vertices that are in the set have already calculated the shortest path. Array D includes paths lengths to all considered vertices. To solve the problem the algorithm needs to access to vertices (V set) and edges weights (matrix C, where C[i, j] store edge weight between vertex i and j).

Finding path is a set of operations:

1. Initialize D with path lengths from start vertex, and add this vertex to S
2. Select vertex u, that u є V – S and D[u] is a minimum
3. For each neighbour q of vertex u set D[q] as min (D[q], D[u] + C[u, q])
4. If V – S is not empty go to step 2

Implementation of this algorithm in the application is almost the same as presented in [1] and [3]. The differences are:

- the algorithm stops when u vertex is a vertex to which path is searched for (this mean that the shortest path was founded)
- also the matrix of predecessors is used, it is filled according to filling D array (this allows to recreate the shortest path vertex by vertex)

Dijkstra's algorithm has complexity $O(V^2)$. This mean that time needed to solve problem is proportional to square of vertices taken into consideration. It is very time consuming in current version of the application. For some paths it can take on minute or even more (in actual version the entire map have more than 2700 of vertices). It is the fastest solution for such kind of problems but it is too slow. I made some observations which can be implemented to make the finding path process faster. It is easy to see that when we look from a path from A to B in typical city we should look not at all possible paths but try to take some subset from point set and then find a path in this subset. It is natural that when we are going from A to B we start to go in the direction to point B and do not go back. This feature is unique for this kind of graph. Figure 22 presents the idea.

[3] The proof for Dijsktra's algorithm can be found in [3] in section 25.10

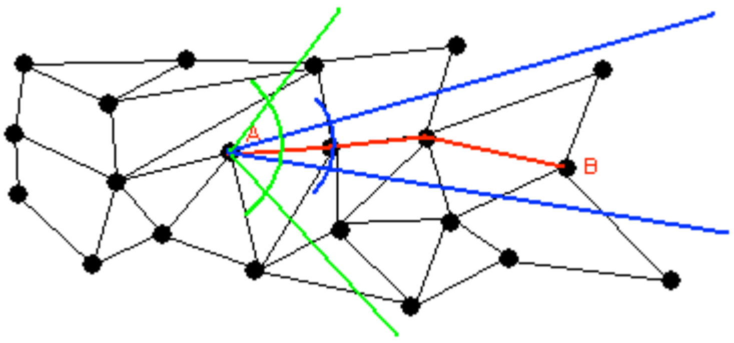


Figure 22. Improving Dijkstra's algorithm

In the presented example looking for the path from A to B do not need to check all possible connections between those points in the graph. If some subset will be chose (between blue lines or green lines) and if in this subset of vertices algorithm can find solution it is the best solution. The motivation for this is that the graph represents the map and connections between points are shortest. If X and Y are neighbours and there is a connection between them, it is the shortest path. In this case if path from A to B will be found in subset it still in harmony with Dijkstra's algorithm proof.

This idea is used to make path finding faster. The 90^O angle is used. It was chosen as a result of few experiments with other angles. If the path will not be found then the algorithm restart and start from beginning looking in entire graph. This angle allows to fast finding the path in selected direction with only few restarts in specific situations (for example when point is on island and we must go back and then take a good direction). Angles bigger than 90^O need more time to find path but also do not restart so often, angles smaller are faster but the have a lot of restarts and in consequence the whole process is longer. Because of that selected angle is compromise which allow find the path in shorter time than without implementing presented observation.

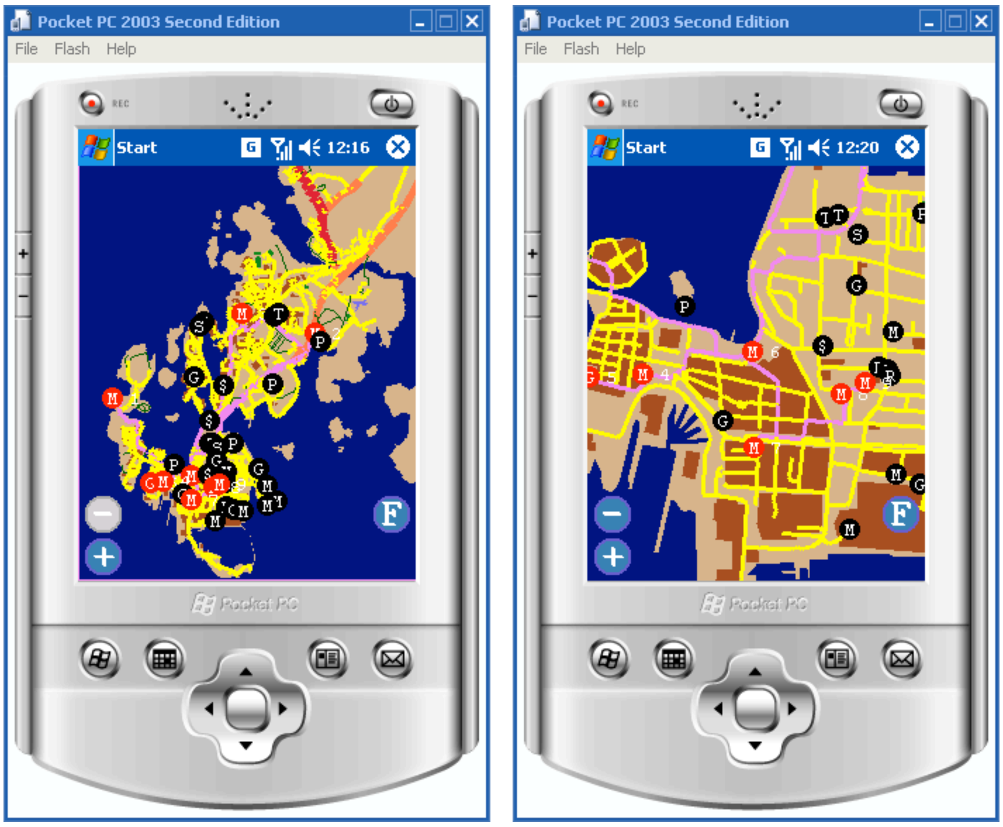
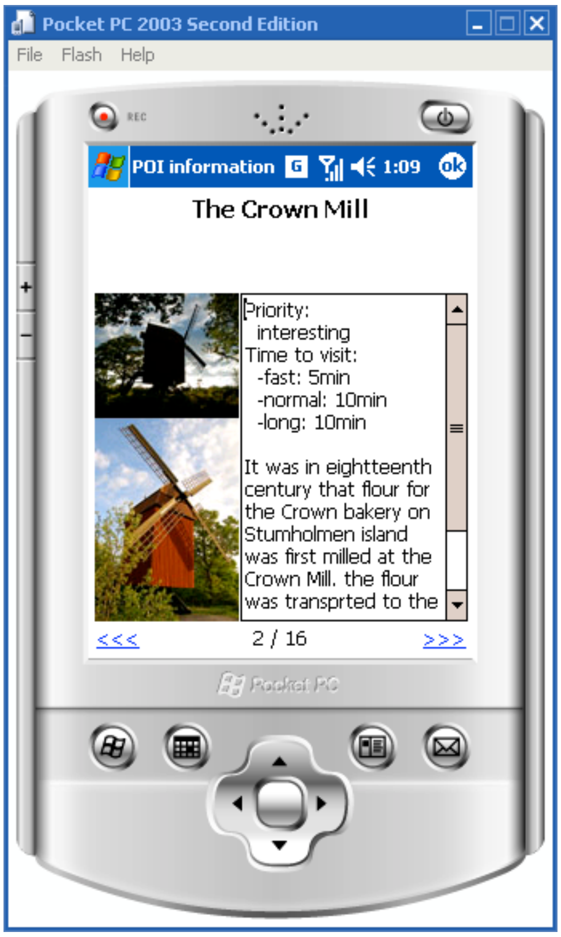


Figure 23. Application screen with found trip

The path should be found with accordance of transport type chose by user. This is the most important for trips where transport is set as vehicle. For points that do not have access by car finding path on map algorithm found the closer point which can be reached by vehicle. This point is later used to find path using constraint for vehicles.

Figure 243 shows the screen with found trip on map. Changing zoom allow to see whole path. Shortest path between each two POIs are market using violet colour.

User can also browse path and read descriptions about POIs which can be visited. This situation is presented on Figure 24.



Priority:
interesting
Time to visit:
-fast: 5min
-normal: 10min
-long: 10min

It was in eightteenth century that flour for the Crown bakery on Stumholmen island was first milled at the Crown Mill. the flour was transprted to the

Figure 25. POI Information window.

5.5 Constraints and requirements

Algorithms for finding paths in graphs are time and resource consuming. From this point of view there are some non-functional requirements to be addressed for the designed application.

1. Fit in memory – for existing devices it is not a problem to allow application use few megabytes of RAM memory, also data structures were designed to be small as possible but allowing for fast data processing.
2. Soft real time – results should be produced in not more than one second after the user specify criteria of the trip. It was very hard to implement. I do not fulfill (finding path on map sometimes take a lot of time).

Developing an application which brings some trip proposition on time was very important for me. I test a few applications (chapter 2) and they are rather fast. It is very important to produce results on time because sometimes users want only to check something and see other alternatives. As it was described in 5.1 I made few assumptions that allow fast calculating a trips and that finds POIs that should be visited. This allows to bring good enough solution just after the user has specified the criteria. Then he can look what objects are on the trip, what is the approximated trip length and time for it. User can change the criteria or if some trip looks good for him can decide to display it on map.

6 FUTURE WORK

The application was implemented and it delivers satisfying solutions and suggestions in short time. To decide if it is good software or just it was a nice idea without something more a small summary must be done. There are potential places for improvements and some work that can be done in future. In this chapter I would like to look closer results produced by the application and try to evaluate them. After that I describe what can be added to the application and what can be improved.

6.1 Evaluating the map building process

There is one important thing which should be taken into consideration while evaluating my work. While building the map I used a raster map which is not for commercial use. To make the commercial application which can be sold and brings profits some maps must be bought. It is not important what kind of a map it will be. Rights to vector or raster maps for commercial use are needed to produce the software which can bring benefits.

Buying rights to a vector map can be an easy solution but as it was described into chapter 2 there are some traps. Buying rights to a raster map can be cheaper but need a lot of work and for commercial use the application that automates vectorization process should be made. It should not be very difficult because if we look at maps that are available in the internet the conclusion is that they are rather simply. Each object has a colour according to its type. In most of cases streets are yellow, green places are green and so on. When maps from one company will be used each objects can be describe as possible colours and then application will be able to find them and describe using points and lines (vector map). But it is not the whole process. The next step is to give names for objects. This probably needs interaction with a person that can do it. Of course this process can also be automated but text recognition will be needed to read street names and objects names from raster map. When text recognition mechanism will be used still somebody should check if it goes right and check for bugs. Another problem is that sometimes determining street name on map can be confusing for people.

I build the map and in my opinion it is good, but time needed to build it was long and need to be reduced. This map can be used only to my personal use only for this thesis. In future those two gaps should be removed if this software becomes commercial.

6.2 Evaluating the user interface and the performance

The main application window was designed without any standard control. Intelligent Tourist Information System is software which should be easy in use. Navigation on the map is very simple, the user can use buttons that each mobile device has to scroll on the map and also he can use drag & drop technology (click on map and just move). Zoom in and zoom out functionalities are also available. Round buttons "+" and "– " that are drawn as graphic allow doing this. Those buttons can be disabled if action performed by them is not possible in some situation (look at Figure 23 and zoom out button "–").

The planning path window is designed accordance with Microsoft Windows guidelines. Elements are grouped and aligned. Each screen does not have lot of components so it is easy to use and for the user it is fast determine where he is and what can he do.

If the application is doing some action and can not react on the user events a progress bar is displayed. It informs the user about progress of action that is actually performed. Some of those calculations take a lot of time but if the user can see the progress bar that informs him about actually state it is not so irritating. This part of the user interface is strongly connected with performance and for some application activities the performance should be improved. The map drawing process is one of the most frequently action which will be performed by the application. It takes not more than 2 seconds in the worst case (maximal zoom out). It is acceptable but I think it should be improved. Improving can be done by making another memory bitmap and drawing on them in another thread when application detect that redraw map action can be needed in short time. The most time consuming action is finding a path on the map and this process can also be improved.

6.3 Results produced by application

In chapter 5 the finding path process was described. Also some explanation about finding solution and proposition for trips and time aspects of this process can be found there. The results are produced satisfying me results in very short time. The other criterion for results is something what is not measurable. It is very difficult to determine if suggestions generated by the application are good or bad. They are fulfilling for me but for other people they can not good. In my opinion paths which is drawn on map is good and it can be acceptable. The trip proposed by the application is only a base plan and it can be and should be changed by the user. From this point of view I think that I fulfill this requirement. Trips produced by application consist of most famous POIs according to their priorities. Proposed solution has estimated time and calculated length and it fit in user criteria. The trips are only good suggestions[4].

6.4 Future work

The prototype application is working and it is fully functional, but it needs some work to make it better. There are few things that I want to improve. I will describe them in sequence that corresponds to their priorities.

6.4.1 Improving the path “smoothing” algorithm

Because “smoothing” algorithm do not produces the shortest trips the improvement is needed. As I said Floyd-Warshall’s algorithm can be used to solve this problem. It is very time consuming and not fast enough for mobile devices algorithm so it should be taken into consideration and tested if using it allows to fit in time constraint. Another solution is to analyze paths produced by implemented algorithm and try to improve it. This choice will be probably preferred by me in future work. The motivation for this is that POIs in the path are almost in order and they should only be corrected. The implemented algorithm concentrates on two points and tries to switch them. The improvement should try to operate not only on neighbours but on POIs that are separated by two or three others. If the improvement will be implemented it should be tested if it still fit in time and is fast enough. In this case the new algorithm can be more complex than Floyd-Warshall.

6.4.2 Symbian OS application

The Symbian OS application was not completed and it is still in developing phase. My Symbian device can only store files that are not bigger than 0,5MB. Currently map of Karlskrona has 470kB and application is something about 50kB so it is too big for my device. It is not fully implemented and tested. I can only run it only in emulator. Latest

[4]Look to 5.2 part and on Figure 21 it will be very easy for user change something in trip if he is not satisfied.

devices have more disk space and have more operational memory so converting the application to latest Symbian OS version can allow to run the application on a phone. Theoretically latest devices are fast as palm devices so they probably can handle the application with the same performance as Windows Mobile devices.

6.4.3 Map rotation

Map rotation is not implemented in the application. Matrix of transformation is used to converts points from map to screen and this is the point where rotation ability should be added. It does not need a lot of implementation. Map rotation will be a good feature that in some cases can make application more usable. The other motivation to add this feature to the application is that all the tested applications (chapter 2) have this functionality and I it is very often used people. It is more natural when on right side of street we have the same in place where we exactly are and on the map on which we are looking on. For a commercial product this functionality will be important as zooming so it needs to be implemented.

6.4.4 Searching objects mechanism

Functionality that allow search for POIs, streets and areas should be added to allows finding any object using its name or some part of description. It is very important to give the user ability to do this. It is very easy to reach this requirement because all needed data are stored in map structures (POIs database).

6.4.5 Using GPS receiver

The map has matched geographical positions and they can be used with GPS receivers. The application does not use a GPS receiver and do not help user to navigate while the trip is in progress. The functionality that shows actually position of device should not be difficult to implement. More complex to implement can be real time navigation and a voice that will lead the user while trip is in progress.

Implementing this functionality can be also complicated because there is not defined standard for communication with GPS receivers in Windows Mobile 2003. Only latest versions of this system have special API for this. The situation is more complicated on Symbian devices, because only few of them have GPS and each of them has different API.

7 CONCLUSIONS

This chapter of the thesis concentrates on summarizing work that has been done in thesis. Answers for research questions are also presented.

While doing the thesis research I build the vector map of Karlskrona. It contains most of streets in city with their names, islands, buildings, parks, lakes and so on. An application to build map was also designed. The application was build on palm device and in the near future it is not completed for mobile phones with Symbian OS.

The basic research question was *"Is it possible to build such system and to get satisfying results?"*. The application that was build can be a proof that it is possible. Mobile devices can handle such kind of application and can be used to find and suggest paths for trip. The second part of question is not so easy to answer. Output produced by software is some proposition for trip. As it was presented in chapter 5 there are still some weaknesses that can be corrected. Calculated paths are good suggestions and can be used as a base plan for the trip. They were presented in section 6.4 of this thesis.

What are weaknesses of existing GPS systems and applications for path finding?

Existing path finding applications were presented in chapter 2 and at the end some weaknesses and strength points of those applications are presented. As it was said the main gap is that those application allow only finding path between points that user add to trip and they do not come up with suggestion.

What are user expectations to a tourist information system? Can they be fulfilled using existing devices?

User expectations were presented at beginning of this thesis. People would like to have all information gathered in one place, and be able to access them any time. Mobile devices give those abilities (chapter 3). Portable devices are nice gadgets which can provide almost any functionality that is possible on typical desktop computer. They have small screen but most of people prefer to have mobility and can accept this weakness.

Which algorithm for path-finding will be the best to fulfill user requirements and needs while travelling (different criteria, maybe two or more algorithms should be used for best search results)?

Chapter 5 has an answer to this question. The designed solution was presented there. There is no one good algorithm to solve problem of finding trip. This part of the master thesis talks about observations and conclusions that I made. Solution that is provided in the application is also presented. At least some improvements that can be implemented are presented in part 6.4 of this document.

How to organize data for most optimal memory consumption?

The chapter 4 describes this problem. Constraints of mobile devices are taken into consideration and compared with applications needs. In this part solution for problem with data structure and drawing map are presented.

All of the expected outcomes have been done. The algorithm for finding the path has been designed and tested. Methods for data gathering and storage were also presented. At least the standalone application for portable device was implemented and will be presented. It works but I still see some place for improvements. The fact is that if it will be a commercial application it always will be evaluated, changed and improved. When we look on the application like on prototype I think that it is working well and it satisfy.

LIST OF FIGURES

BIBLIOGRAPHY

[1] Aho Alfred V., Hopcroft John E., Ullman Jeffrey D. – *Data Structures and Algorithms* Bell Telephone Laboratories (1983)

[2] Babin S. – *Developing Software for Symbian OS. An Introduction to Creating Smartphone* John Wiley & Sons (2006)

[3] Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Stein Clifford – *Introduction to Algorithms, Second Edition* The MIT Press (2001)

[4] Digia Inc. – *Programming for the Series 60 Platform and Symbian OS* John Wiley & Sons (2003)

[5] Grewal Mohinder S, Weill Lawrence R., Andrews Angus P.– *Global Positioning Systems, Inertial Navigation and Integration* John Wiley & Sons (2001)

[6] Praca zbiorowa – *System nawigacyjny GALILEO. Aspekty strategiczne, naukowe i techniczne* Wydawnictwa Komunikacji i Łączności WKŁ (2006)

[7] Sanders P. – *Symbian OS – the Open Mobile Phone Operating System* (article from http://www.symbian.com, *last visited January in 2007*)

[8] Wirth Niklaus – *Algorytmy + Struktury Danych = Programy* (org title *Algorithms + Data Structures = Programs)* Wydawnictwa Naukowo Techniczne (2002)

[9] http://ec.europa.eu/dgs/energy_transport/galileo/, *last visited in June 2007*

[10] http://en.wikipedia.org/wiki/SiRFstar_III, *last visited in May 2007*

[11] http://en.wikipedia.org/wiki/Windows_Mobile, *last visited in February 2006*

[12] http://en.wikipedia.org/wiki/Windows_Mobile, *last visited in February 2006*

[13] http://opie.handhelds.org/cgi-bin/moin.cgi/WhatIsOpie, *last visited in February 2006*

[14] http://pl.wikipedia.org/wiki/Palm_OS, http://en.wikipedia.org/wiki/Palm_OS, *last visited in February 2006*

[15] http://www.trolltech.com/products/qtopia/, *last visited in February 2006*