jamk.fi

Navigation application in the building via two-dimensional barcode recognition

Michal Banik

Bachelor's Thesis
May 2014

Degree Programme in Software Engineering
Technology, communication and transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s) Banik, Michal	Type of publication Bachelor´s Thesis	Date 14. 05. 2014
	Pages 62	Language English
		Permission for web publication (X)
Title Navigation application in building via two-dimensional barcode recognition		
Degree Programme Software Engineering		
Tutor(s) Mieskolainen Matti		
Assigned by University of Žilina		

Abstract

The aim of this thesis was to create a navigation application working on the principle of recognizing two-dimensional graphical codes. This application should support the creation and editing of two-dimensional maps of the building and assign specific information to the objects on the map. The thesis describes the basic algorithms usable for indoor navigation, followed by a brief description of available techniques and methods of navigation. The description of QR codes used in this work to locate the position of the building is included as well.

In the design part the study, more detailed description requirements are added and the concept of the application implementation is described. The model building used in the study presents the building as an object with floors and rooms. It distinguishes two types of rooms, the plain rooms on one floor and the rooms connecting the floors. The implementation is divided into map editing section and navigation section. Two applications have been created based on the requirements. FriBasicMapEditor, as editing application, provides functions for creating and editing two-dimensional maps of the building and matching QR code to objects in maps. The navigation application, FriNavigate uses a map created by FriBasicMapEditor application and provides for user the navigation functions in the building described by loaded map. The navigation is based on the recognition of generated images of QR codes.

The application can be used to navigate in any standard building or used as a basis for more complex, respectively more extensive projects related to navigation in buildings.

Keywords
indoor navigation, QR code, model of building, map editor

Miscellaneous

CONTENTS

FIGURES

TABLES

1 INTRODUCTION

In the Middle Ages, the first mariners used stars to determine their position on their journeys. It could be said that the stars navigated them to areas that they did not know.

It is normal that orientation can be lost when in unknown areas. The more this area is ragged, the more this statement is true. In prehistoric times the humans mapped the land where they lived. They found food, pastures and also they needed to know how they can get back home. The orientation is made easier using some devices, from the first maps in the rock caves through compasses and to electronic navigation systems used today.

The idea of navigation is not only to find some path from one point to another point, because some connection of two points exist almost every time on the real map, in the real world. The principle of navigation consists in optimizing the length of the found path to make it as short as possible. This results in other positives that are, for example, saving transition time from one point to another or saving costs for transit between these points. One option how to optimize this distance is to use algorithms from a mathematical discipline called graph theory, i.e. specific algorithms for calculating the shortest path between two given points. Therefore, the beginning of this thesis introduces the basic algorithms for finding the shortest path.

To use the algorithms is not sufficient. Before applying the algorithms it is needed to know the position where one is and also the position where one wants to find the path to. If using a paper map somewhere outside mostly it is not problem. The problem occurs, however, when at an unknown place and there is no knowing of how to orientate at this place, nor is there knowledge of the position which could be given to the algorithm as a starting point. There are many techniques, devices and systems to locate the position some of which are to be described in the next part.

The main aim of this thesis was to create an application which would facilitate visitor orientation in an unknown building and provide navigation to a given destination. Complicated electronic systems for localization consume a great deal of funds, therefore they are not too effective mainly in terms of finance. The application which is the content of this thesis uses recognizing two dimensional graphic codes for localization. For scanning these codes it is sufficient to have some device for recording images, e. g. camera. The navigation application demands only to transfer the scanned image of graphic code from the recording device. This way, the navigation is financially and technologically very convenient. The system does not need any radio link for communication and thus it works very quickly, without connection drops and without interferences with other devices. Because the navigation is based on recognizing two dimensional codes, the thesis work contains also the basic description of the selected 2D code – description of QR code.

2 ALGORITHMS FOR FINDING THE SHORTEST PATH

Algorithms for finding the shortest paths are in practice highly desired. These are mainly used for optimization tasks of different transport networks, but they are also and often part of other algorithms.

2.1 The basic algorithm

The basic algorithm seeks the oriented path from a fixed vertex u to a reachable vertex v in positively valued and directed graph (i.e. a digraph). The complexity of the algorithm is $O(n^3)$.

Almost all algorithms for finding the shortest paths in graphs with the fixed starting point use the **principle of the basic algorithm**. This principle is based on assigning two variables to each vertex, t and x. With these variables the steps of the algorithm are performed, which are described below. The first variable $t(i)$ describes the last found shortest path to this vertex from the starting vertex, the second variable $x(i)$ is the previous vertex of the found path.

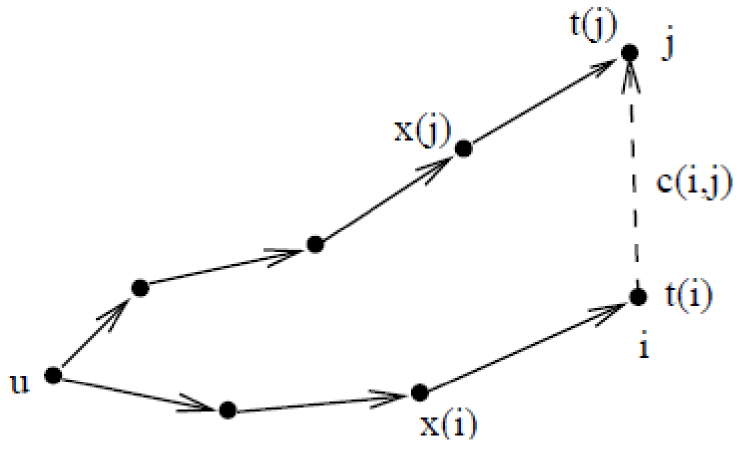


Figure 1. Principle of the shortest path algorithm finding (Palúch 2008, 72)

During calculation the condition is applied: if vertexes i and j are the incident and if is it true that $t(j) > t(i) + c(i,j)$, where $c(i,j)$ is the value of edge starts in vertex i and ends in vertex j, then the last found path to the vertex i with length $t(i)$ extended by the edge from i to j with length $c(i,j)$, is shorter than the previously found path to the vertex j. In the case when this condition is true variable $t(i)$ and $x(i)$ is overwritten as follows: $t(j) = t(i) + c(i,j)$ and $x(j) = i$. This part is also called the **edge relaxation**.

The exact steps of the basic algorithm for finding the shortest path are as follows.

- **Step 1.** Initialization. Assign two variables $t(i)$ and $x(i)$ to each vertex $i \in V$, ($t(i)$ describes an upper estimate of the last found shortest path from the vertex u to the vertex i and, $x(i)$ describes the previous vertex of this path, V is the set of all vertices). Set the variables $t(u) \coloneqq 0$, $t(i) \coloneqq \infty$ for $i \in V, i \neq u$ and $x(i) \coloneqq 0$ for all $i \in V$.
- **Step 2**. Find out whether there is an oriented edge from i to j, $i, j \in H$ (H is the set of all edges of the graph) such that the following applies $t(j) > t(i) + c(i,j)$, where the $c(i,j)$ is the value of edge starts in vertex i and ends in vertex j. If such an edge exists, set $t(j) \coloneqq t(i) + c(i,j)$, $x(j) \coloneqq i$ and repeat step 2
- **Step 3.** If there is no edge to satisfy the condition of step 2, create back the shortest path from vertex u to vertex i using the variables $x(i), i \in V$. The shortest path will be: $i, x(i), x(x(i)), x(x(x(i))), \dots, u$ (displayed from ending vertex to initial vertex).
- END

After the algorithm ends, the shortest path is known from vertex u to any vertex $i \in V$, the length of which is in the variable $t(i)$. If $t(i) = \infty$, the vertex i is not reachable from vertex u. (Palúch 2008, 73)

2.2 Dijkstra's algorithm

Dijkstra's algorithm is one of the most used algorithms for finding the shortest path between two vertices. The basic algorithm is able to find a path between the start and some other reachable vertex. Dijkstra's algorithm solves finding only one shortest path connecting two vertices of an oriented graph with non-negative valued edges. This gives the Dijkstra’s algorithm **less complexity** $O(n^2)$. In case when the path from starting vertex to the concrete only one ending vertex needs to be known, then this algorithm is better, however the algorithm fails in case when the graph contains a negative valued edge. (Matulová 2009, 25)

As hinted above, Dijkstra's algorithm is based on the basic algorithm. It assigns the same variables to vertices $t(i)$ and $x(i)$ but with one difference, two states of the variable $t(i)$ are recognized. This variable can be set as final, and then it is no longer able to change or can be sat as temporary. (Le, Saragas & Webb 2009, 29)

The algorithm consists of the following steps (Palúch 2008, 80):
(The shortest path between vertices u and v is looked for)

- **Step 1.** Initialization. Set each vertex's variables $t(i)$ and $x(i)$. Variables $t(i)$ will have two types namely the temporary or final. Set $t(u) \coloneqq 0$, $t(i) \coloneqq \infty$ for $i \in V, i \neq u$ and $x(i) \coloneqq 0$ for all $i \in V$. Select the pilot vertex $r \coloneqq u$ and variable $t(u)$ (resp. $t(r)$) as final, the remaining variables $t(i)$ set as temporary.
- **Step 2.** If $r = v$, END. If $t(v) < \infty$, the variable $t(v)$ indicates the shortest path from u to v. The path can be found back from variables $x(i)$. If $r \neq v$, for all edges from r to j, where the vertex j is set as temporary, check condition $t(j) > t(i) + c(i,j)$. If the condition is true set $t(j) \coloneqq t(i) + c(i,j)$, $x(j) \coloneqq i$ and changed variable keep set as temporary.
- **Step 3.** From all vertices marked as temporary find one vertex with the lowest value of variable $t(i)$ and set this variable as final and choose it as pilot vertex $r \coloneqq i$. Continue to Step 2 (If there are multiple vertices with the lowest value of $t(i)$ select one of these vertices, the other will be used in next iteration.)

The algorithm can be easily remade for searching the shortest path from the starting vertex to all other vertices when the condition for terminating is changed to the condition: END when all the variables $t(i)$ are marked as final.
(Palúch 2008, 80) (Le, Saragas & Webb 2009, 29)

2.3 Bellman-Ford's algorithm

The next algorithm to find the shortest path is Bellman-Ford's algorithm. The main advantage of this algorithm is, that unlike the Dijkstra's algorithm, it is able to find the shortest path in the graph with a **negative valued edges** until the graph does not contain a cycle with negative length. (Le, Saragas & Webb 2009, 35)

The specific steps of the algorithm are as follows:

- **Step 1.** Initialization. Assign two variables $t(i)$ and $x(i)$ to each vertex $i \in V$ ($t(i)$, is length of the shortest path found in last iteration and $x(i)$ is the previous vertex of this path). Set variables $t(u) \coloneqq 0$, $t(i) \coloneqq \infty$ for $i \in V, i \neq u$ and $x(i) \coloneqq 0$ for all $i \in V$.
- **Step 2.** Edge relaxation. Apply the condition $t(j) > t(i) + c(i,j)$ for all edges$|V| - 1$ times, where $c(i,j)$ is the value of edge from vertex i to vertex j. If condition is true, set $t(j) \coloneqq t(i) + c(i,j)$, $x(j) \coloneqq i$ and repeat step 2.
- **Step 3.** Relax the edges again. If there will not occur an updating of variables $t(i)$ and $x(i)$ the algorithm ENDS. The shortest path from vertex u to any other vertex in the graph could be constructed using reference variables $x(i)$. If the vertex variable $t(i)$ is ∞, this means that the vertex is not reachable from vertex u. If at least one updating of variables $t(i)$ or $x(i)$ occurred during the relaxation it means that the graph contains a negative cycle and the length of the found path will not be correct.

(Demaine 2011, 1)

This algorithm is used for example in network protocol RIP. The complexity of this algorithm is $O(|V| * |H|)$, where $|V|$ is the number of vertices and $|H|$ the number of pages in the graph (Hedrick 1988, 2).

2.4 A star algorithm

A* algorithm, eventually A star algorithm, finds the shortest path between the starting vertex u and the ending vertex v in undirected graph **using the heuristic method**. It is similar to Dijkstra's algorithm. The difference is in choosing the next vertex. Vertices are selected on the basis of the **function $\boldsymbol{f}$** which is associated to each vertex. This function represents the last found length of the shortest path to this vertex and estimation of distance to the target vertex. Heuristics are used to calculate this estimation, because the real distance to the target vertex is known after the shortest path has been found. The calculation of the estimation can be obtained by various methods. Vertex with a minimum value of function f is marked as visited and the calculation continues from this vertex, this vertex is no longer classified as a candidate for further calculation. Next, during the execution the procedure is like in Dijkstra's algorithm, and thus if any edge exists that optimizes the shortest distance to the vertex currently visited, this vertex will store this better distance and also the vertex which precedes it (aforementioned **relaxation**) (Matulová 2009, 25)

The complexity of this algorithm is $O(|H| + |V| * log|V|)$. Because the solution **is not always** mathematically **optimal**, it is not guaranteed whether the resulting path found is the best solution. (Matulová 2009, 25) (Lester 2005)

2.5 Comparison of algorithms

The following table compares the algorithms, their advantages and disadvantages as well as complexity. (Table 1.)

Table 1. Comparison of algorithms for finding the shortest path (Le, Saragas & Webb 2009, 36)

Algorithm	Advantages	Disadvantages	Complexity
The basic algorithm	simplicity of implementation	slowest	$O(n^3)$
Dijkstra's algorithm	speed, optimality of solution, option to terminate after the shortest path was found	does not work with negative valued edges	$O(n^2)$
Bellman-Ford algorithm	works with negative valued edges	less speed	$O(\|V\| * \|H\|)$
$A*$	high speed	not guaranteed optimal solution	$O(\|H\| + \|V\| * log\|V\|)$

The **fastest** of the above mentioned algorithms are **Dijkstra's** and **A***. If the task requirement would be to work with negative valuation of edges it is necessary to use the Bellman-Ford's algorithm. In case when there is a sensitive task requirements to find the shortest reachable path then it is the best adept Dijkstra's algorithm, since A* algorithm does not guarantee optimality of the found path. (Le, Saragas & Webb 2009, 29)

The given task does not require to work with the negatively valued edges, what confirms the selection Dijkstra's or A* algorithm. Therefore, there is the choice from two algorithms. Because of the ease of implementation Dijkstra’s algorithm was chosen. (Le, Saragas & Webb 2009, 29)

In case of the navigation, an autonomous or semi-autonomous mobile robot in an unfamiliar environment would be good to consider to use **the histogram navigation algorithms**. They are the potential based methods which use histogram grid map area. This grid, which is additive, is updated by robot dynamically according to the distance detection of barriers. There are several methods for this kind of navigation. (Babinec & Vitko 2010, 26)

3 OPTIONS FOR NAVIGATION

Nowadays there are several options available for realizing the navigation in the building. In each case the solution is specified by something else and it includes smaller or larger time and financial costs.

For indoor navigation there is the choice, for example between these technologies and their combinations.

- High sensitive GPS sensor
- Pedometer
- Compass
- Accelerometer
- RFID chips
- Wi-Fi
- Bluetooth
- Camera

3.1 GPS navigation

GPS navigation is designed to be used in **open areas**, roads, or to navigate in the city streets, however, also there is sometimes a problem in large cities to capture the signal from the GPS satellites. (Le, Saragas & Webb 2009, 4)

Due to lack of bad localization in urban area, the navigation in a building needs a very powerful GPS receiver. The alternative is to amplify the GPS signal strength (using a device called GPS Repeater – signal repeater) in every single building and a technology called **A-GPS** (Assisted GPS), which provides additional information through the available communication data links. GPS devices cannot obtain this information themselves due to signal attenuation. (Le, Saragas & Webb 2009, 5)

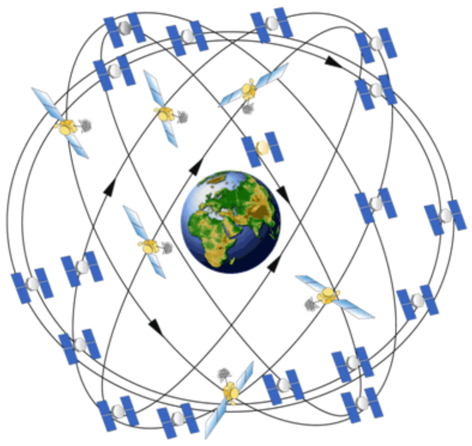


Figure 2. System of GPS satellites (Navrchol.sk 2006)

3.2 Compass and pedometer

Pedometer together with compass, depending on their quality, provides sufficiently accurate detection of the position on the prepared map. In this method of navigation the user determines the **starting point** on the map and the **average length of step**. After this initialization, the device records the direction of movement and travelled distance, which can be given to the user as the current location. (Ausmeier 2011, 8)

3.3 Accelerometer and gyroscope

Using a similar principle as the previous example, also the navigation based on accelerometer and gyroscope works. A gyroscopic device can determine its **orientation in space**, following the comparison with the previous orientation it can determine the directional change. (fyzmatik.pise.cz 2008)

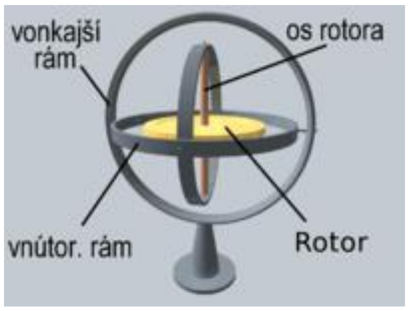


Figure 3. Gyroscope (tahaky-referaty.sk 2008)

Accelerometer measures the acceleration of a device in space and isolates vertical movements from which the **number of steps** can be determined. Then the number of steps is multiplied by the average step length, from which the distance is obtained. By combining these two devices there is another way to determine the current position of the device, and thus the user. (Ausmeier 2011, 8) (Lin, Smith & Wehrle 2011, 1)

3.4 RFID

Radio Frequency Identifier (RFID) is one of the possible ways to determine position in the building. RFID identifier, further RFID tag contains a small **chip** and an **antenna** or also the battery. The chip size can be only 0.5 mm2 and the thickness of the entire tag is so small that it can be inserted into the paper. RFID tags can communicate in a range up to several meters. The cheaper passive RFID tags do not contain a battery. The energy for replying is obtained from received signal. (RFID Tags - Radio Frequency Identification Tags 2009)

RFID tag contains 96-bit information, which is sufficient data capacity for storing any significant practical identifier. (ibid.)

There are different types of RFID tags. Furthermore, there are **four categories** divided by **frequency** in which a tag communicates with the reader, and thus there are LF (125 or 134.2 kHz), HF (13.56 MHz), UHF (868-956 MHz) and microwave (2.45 MHz). (ibid.)

The main categories are active and passive RFID tags. (ibid.)

- **Passive RFID tags**, as mentioned above, have not their own power supplies, but when the signal from the reader is received also a small current is created which is sufficient to send a short reply back to the reader. Thanks to the fact that **no power source** is contained in the passive RFID tags they may have much **smaller size**. These tags can communicate over a distance of 1 centimetre to 6 meters.

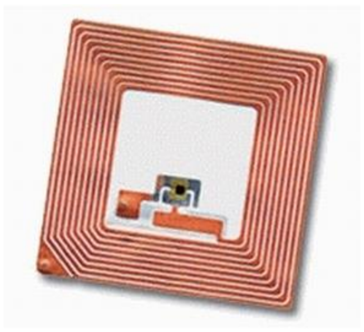


Figure 4. Passive RFID tag

- **Active RFID tags** contain their own power, what gives them a longer time period for the reading and they can also store large amounts of data. Thanks to the source, which is able to keep the tag alive up to 10 years, it is possible to store the information sent by the transmitter. These tags can be read from a longer distance. The size of active RFID tags is the size of a one coin, however, they are **more accurate**, **more reliable** and **more powerful** in difficult conditions than the passive tags.

The major advantages over conventional bar code are listed below (RFID Tags - Radio Frequency Identification Tags 2009)

1. Possibility to communicate over longer distances
2. It has a larger data capacity

In navigation systems, RFID technology can be used for getting the exact location of an object in real time.

3.5 Wi-Fi

The need to use the Wi-Fi technology for indoor navigation was caused due to the increasing use of access points in public and private buildings. The principle is based on a sufficient deployment density of access points of Wi-Fi network. In short, it can be said that the mobile device with Wi-Fi receiver periodically **scans** the surrounding for occurrence of **access points**. (Work 2003, 1)

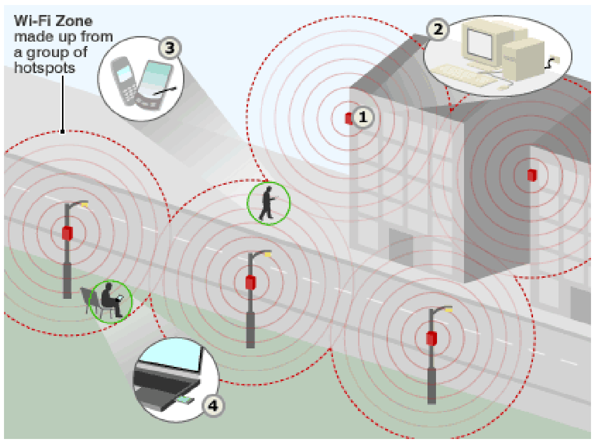


Figure 5. Wi-Fi network example (Image 2012)

In the beginning of navigation the users set their position and the device maps the area where the nearby access points are located. The found access points are stored in the list together with their signal strength. Devices are looking for available access points periodically while the users are walking in the building and compare the signal strength with the last found value, which can determine the position change relative to the navigation start point. Then the application can check whether the user is moving along the route, which should be followed to reach the destination. (Work 2003, 1)

3.6 Bluetooth

Bluetooth navigation is mainly limited by signal **range**, because of which the Bluetooth technology is used mostly for connecting an external navigation device. Using the Bluetooth navigation separately is possible with a **high density** of Bluetooth transmitters. (Ausmeier 2011, 5)

3.7 Video camera or camera

Use of the camera or video camera for navigation in buildings is possible in several ways. One of them is, for example **dynamic image processing**. The second method is based on **capturing** some **reference mark**, which contains location information. By recognizing this information the application can inform the user about the current location. (Ausmeier 2011, 7)

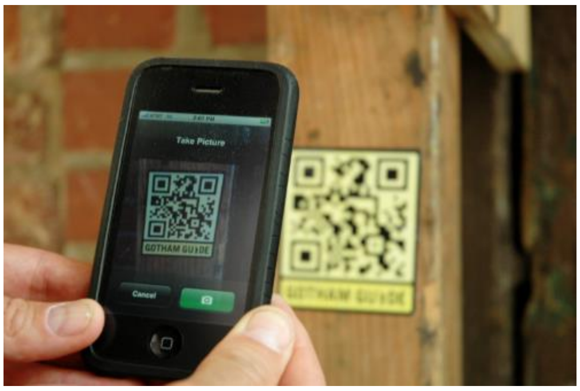


Figure 6. QR code recognition (Image 2009)

The subject of this thesis work will be an easier alternative, the recognition and using reference marks assigned to certain places on the map. Specifically, it will be based on two-dimensional graphical codes. This reference mark clearly defines and identifies the object on the map. It cannot happen that two objects have been assigned to one and the same mark.

This type of navigation system is based on the **existence** of these **reference marks** and thus before the process of navigating starts these tags must be **assigned to the map** objects. In this case, the user does not enter a starting point, but the point is automatically set when the first reference mark is loaded, to which the user arrived. After updating the user's position, this position is displayed on the map. This update occurs with each detected reference mark (different or same if user has returned to this place). This update causes the calculation process of path to the destination specified by the user. Furthermore, each reference mark may include **additional information** about the assigned object, on the basis of its data capacity.

4 QR CODES (QUICK RESPONSE CODES)

The barcodes have become very popular because of their feature to be simple, accurate and fast readable. After the global expansion of barcodes the market demanded codes which could store more information, however, maintaining the same processing speed that is provided in barcodes. The barcodes can store information in one axis, in other words, they are one-dimensional codes. In response to these mentioned requirements, the **two-dimensional codes** began to appear. (KRUPA 2011, 3)

QR Code (Quick Response Code) is a kind of two-dimensional graphical code (2D barcode), developed by DENSO WAVE Inc. QR code unlike the barcode contains information stored in the **vertical** and in the **horizontal axis** (two-dimensional), therefore it is able to store several times more information. (Denso Wave 2011)

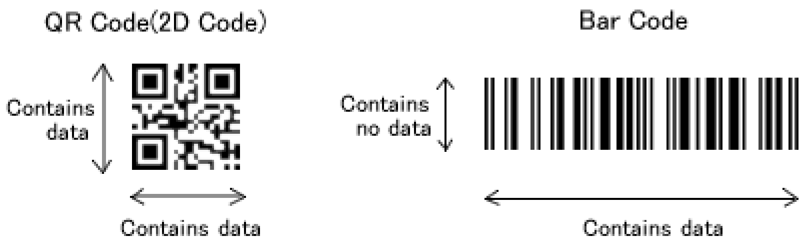


Figure 7. QR code and bar code comparison (Denso Wave 2011)

Except QR codes, also other two-dimensional codes have been developed. Typical two-dimensional codes are shown in Table 2.

4.1 Properties of QR codes

The QR codes provide a **greater data range** compared with conventional bar codes. As it is shown in Table 2, this feature is maintained also in comparison with other two-dimensional codes. QR codes can store up to 7089 digits or 4296 digits combined with letters - alphanumeric characters. (Denso Wave 2011)

QR codes can be displayed on a **smaller area** than barcodes. Because the QR code contains the information in two axes, it reduces the area which will be displayed (or printed), in some cases up to 10 % of the conventional bar code with the same information value. For small amount of information the reduced version of QR code called Micro QR Code can be used. (ibid.)

QR codes have the ability to repair the damaged part of its area. **Error correction** is based on the Reed - Solomon algorithm. It is an efficient algorithm, which has the capability of correcting clusters of errors. With this algorithm the detection part of the code can be simply allocated and it can be used for application defined by standards CSN EN 50159-1 and EN 50159-2. (Franeková M. 2003, 3)

The following table compares different types of two dimensional graphic codes. It shows company which developed the specific code, capacity limitation, advantages and organizations which standardize specific code. (Table 2.)

Table 2. Two-dimensional codes comparison (Denso Wave 2011)

		QR code	PDF417	DataMatrix	Maxi code
Code example					
Developed (country)		DENSO (Japan)	Symbol Technologies (USA)	RVSI Acuity CiMatrix (USA)	UPS (USA)
Type		matrix	composite bar code	matrix	matrix
Maximum capacity	Numeric	7,089	2,710	3,116	138
	Alphanumeric	4,296	1,850	2,355	93
	Binary/byte	2,953	1,018	1,556	-
	Kanji/kana	1,817	554	778	-
Main advantages		large capacity, small size after print, high reading speed	large capacity	Small size after print, large capacity	High reading speed
Standardized		AIM International, JIS, ISO	AIM International, ISO	AIM International, ISO	AIM International, ISO

Depending on the level of error correction that was used when the code was generating, the data **up to 30%** of the total area can be restored. Together with increasing level of error correction the size of the code is also increased. The available levels are shown in Table 3 together with the value of bug fixes. The most common level is M, i.e. 15% of information stored in the code can be damaged without being lost.

With three detection patterns at the corners the QR code has not reduced the readability even in the case of rotated code. The code can be processed with sufficient speed regardless of the angle of rotation in the range $< 0^\circ;\ 360^\circ >$.

The following table shows all levels of QR error correction ability (Table 3.)

Table 3. Levels of error correction ability (Denso Wave 2011)

Error correction ability	
level L	around 7%
level M	around 15%
level Q	around 25%
level H	around 30%

QR codes support a function called **structured linking**. This means that the code can be divided into several smaller QR codes and the reader will reconstruct the code into a single unit when all parts are loaded, if the reader or decoder supports this function. This function is used when there is a limited size of the presentation device. QR codes can be divided **up to 16 parts**, which then are shown in a row. In Figure 8 this function can be seen applied. After the decoding process is applied on both top image and small images at bottom, the information read from the code at the top will be the same as the information from 4 small QR codes at the bottom. (Denso Wave 2011)

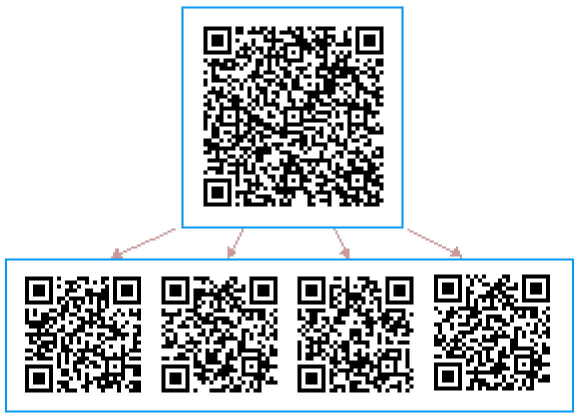


Figure 8. Structured linking of QR codes (Denso Wave 2011)

As symbology developed in Japan, QR codes provide support for encoding **Kanji** character set. In case of use Japanese Kanji characters one character is encoded into thirteen bits. QR technology can store approximately 20% more data compared to the other two-dimensional codes. (Denso Wave 2011)

4.2 Standardization

QR code was introduced in 1997 as a standard for automatic identification as AIM International Standard (AIM-IDS 97/01). Later in 1999, it was registered as Japanese Industrial Standard (JIS-X0510). Also, in the same year it was accepted as a standard two-dimensional symbol usable for EDI transaction forms in the Japanese automotive industry (JAMA-EIE001). Based on this standardization it has been nominated as an ISO / IEC JTC 1 SC 31 standard. This standard was approved in 2000, as an **international standard ISO / IEC** (ISO/IEC 2000).

As a technology able to handle different character sets it has been adopted in other "eastern" countries too. In 2000, it was accepted as a Chinese National Standard (GB / T 18284). Later in 2002 it was accepted as a Korean National Standard (KS X ISO / IEC 18004) and as a Vietnamese national standard in 2003. (QRBCN 2008, 68)

4.3 Structure of the QR code

QR Code is a graphical square shaped code. The actual structure is the regular grid divided into **data part** and **functional part**. (Denso Wave 2011)

The size of QR code depends on the version of code which corresponds to the data capacity, further it depends on the type of character, the level of error correction and the size of the module, which depends on the power of the printing device and reading devices. (Denso Wave 2011)

One **module** represents one bit coded and each module is a square with side length of 4 or more points. All modules in one QR code have the same size. (Denso Wave 2011)

The detailed structure of the QR code is shown in Figure 9.

QR codes were developed especially for labelling and tracking parts in the automotive industry, however, in expanding the codes to the other fields they were begun to be used for encoding different contact or identification information. Because of the high reading speed they were also expanded to the field of mobile communication. When the mobile device loads the decoding application only the camera is needed for decoding code, the device can read the code anywhere, anytime. Therefore the recognition of various data formats stored in QR codes was implemented into the decoding application. Many application can nowadays recognize many formats, for example URI and URL format (email address, phone number, Web address), contact information, SMS and MMS, geographical location, information about events and thus the usability of the QR codes is more accessible for more people. (Denso Wave 2011)

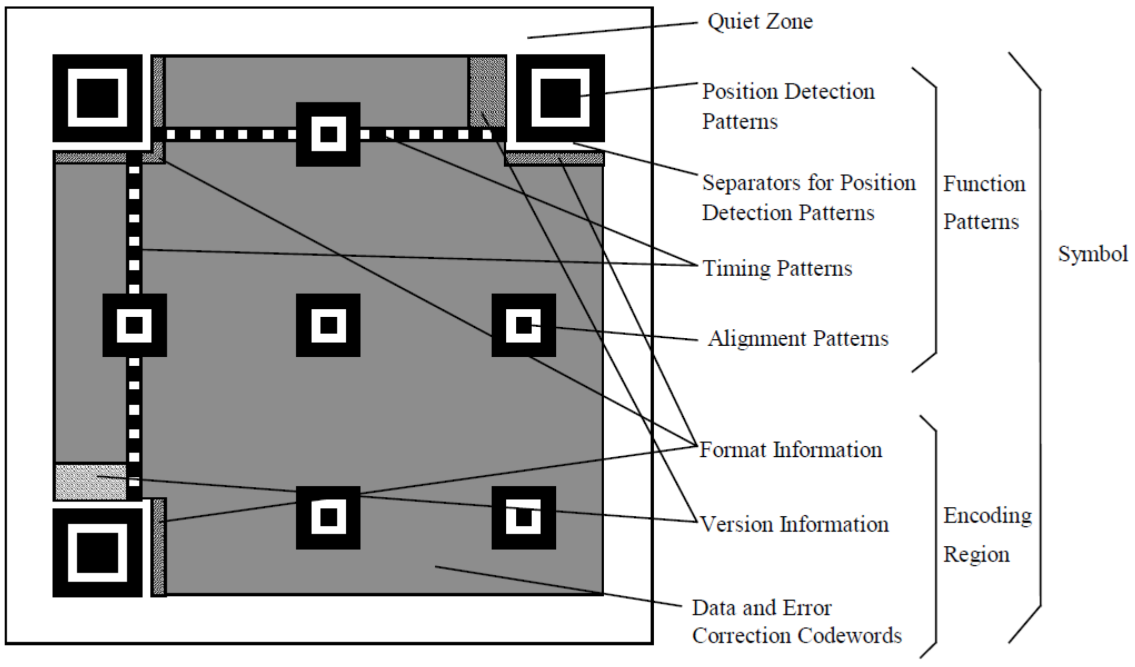


Figure 9. QR code structure (ISO/IEC 2000)

The **data section** contains the actual encoded data words, words inserted by the aforementioned Reed-Solomon algorithm for fixing code, version information and format. (ibid.)

Three positioning **detection patterns** in corners of code provide the possibility to read the code at any angle of rotation.

The one part of the QR code is also “**quiet zone**” (according to Figure 9) or separation zone (KRUPA 2011, 15), with four or more modules on each side of code.

QR codes have defined **40 different versions**. Each version differs in the number of modules and thus each is limited to a certain amount and type of data and the error correction level. Version 1 contains a 21x21 matrix of modules, each additional version contains 4 more modules on each side up to version 40, which is a matrix consisting of 177x177 modules. The accurate values of the data limitation based on version of code are listed in **Appendix 2**. (Denso Wave 2011)

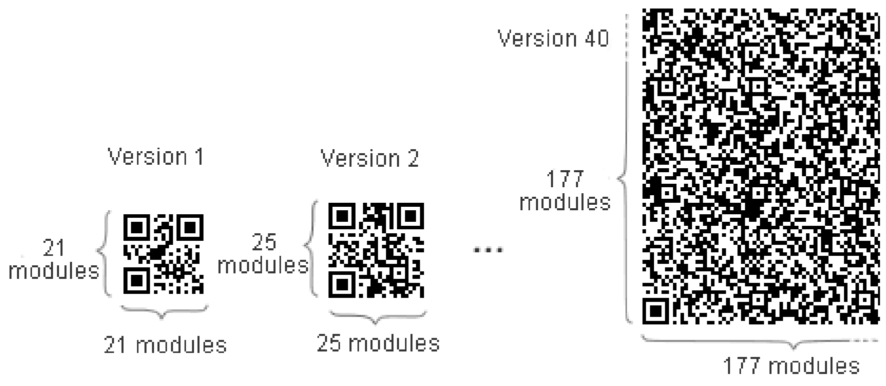


Figure 10. Version example (Denso Wave 2011)

5 DESIGN AND IMPLEMENTATION

5.1 Thesis assignment

The assignment of the thesis was to create the application for localization on the created map. The navigation is based on two dimensional code recognition. The application should contains the functions for generating, recognizing and printing of two dimensional codes. Furthermore, the application should provide environment for creating and editing two dimensional map of building. Assigning the generated codes with other information to the map should be part of the map editing function. As a core application function is considered the navigation from one place to another on the map.

5.2 Two dimensional code selection

Because of the specification availability, libraries for graphical codes processing for different programing languages and possibility to assign additional information to the objects on the map, the **QR codes** were selected as a main representative of two-dimensional codes used for the application.

The comparison and advantages over the other graphical codes can be found in chapter 4.

It will be the best to use external library for QR codes processing which was already created because of the time available.

5.3 Programing language selection

The language for the implementation can be selected between some possible alternatives such as C++, Object Pascal or Delphi, or Java. Because the author has the greatest knowledge of programming in Java language, the **Java** was chosen.

The Java, originally developed by Sun Microsystems, which is now part of Oracle Corporation, has a very similar syntax like C language. Unlike the C language the source code is not compiled directly into machine code, but in the code called byte code, which is then translated by the JVM (Java Virtual Machine). Thanks to the JVM exists for every today most spread operating system, the applications written in Java language can be run on almost any platform. There is another advantage of this language for this work namely that the applications for the Android, as an operation system for mobile devices are based also on Java language, so the application porting for using in mobile phones with this operation system will be that much easier. (Android Developers Portal n.d.)

5.4 Basic layout of the design

Based on the assignment of the work the conclusion was reached that it would be suitable to solve the requirements as a development of two applications. So this study was divided into two parts:

- **Navigation application** (FriNavigate) - includes processing of created maps, updating the position based on the recognition of two-dimensional printed codes (QR codes) and navigating from one place to another on the map.
- **Map editor application** (FriBasicMapEditor) - contains functions for creating and editing maps, functions for generating QR codes and assigning them to the place on the map. Output from this application is the map model as an object file which can be then read with FriNavigate application.

There is a library used for processing the two dimensional codes in the applications also, namely the processing of QR codes is used. It is a **library ZXing** (ZXing library 2007).

ZXing is an open source platform for processing two dimensional bar codes developed in Java language. This library has also a partial ports to other languages such as. C + +, C #, Objective C, Actionscript. The library is able to encode and decode the following graphical codes: (ZXing library 2007)

- UPC-A and UPC-E
- EAN-8 and EAN-13
- Code 39
- Code 93
- Code 128
- ITF
- Codabar
- RSS-14 (all variants)
- QR Code
- Data Matrix
- Aztec (beta quality)
- PDF 417 (alpha quality)

The figure below describes the basic design how the application works.

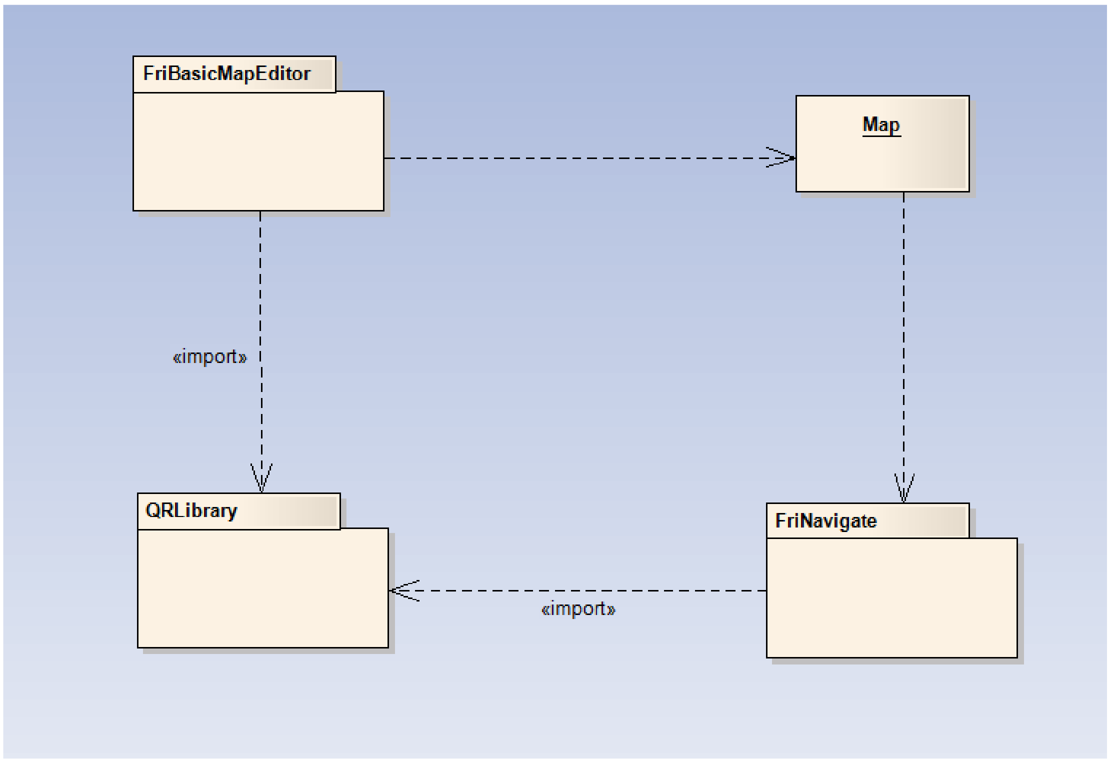


Figure 11. Basic design

5.5 Building mapping (map format)

For processing of floor plans of entire building in the application, it is necessary to change this building to some **model**. It is necessary to identify single floors and individual **rooms** on a particular **floor**, when navigating inside building. But buildings contain also places that cannot be really considered as rooms, for example **stairs** and **elevators**. The model for navigating can describe the place like that also as a room, but with other properties.

Of the logical considerations it is clear that every room in the building is unique and cannot be defined in two different places at the same time. This limitation means that each instance of a room can only be on one floor and also may occur only once in a building. This restriction after a small modification does not apply for elevators and stairways; however, still, for these "kinds of rooms" it applies that only one instance of this object may be on a single floor.

Another element which should be included in the model is an object called **door**. The doors are more or less the connecting elements between two rooms. Therefore, for the solid model without any interruption in routes, it is necessary that a door should "know" what rooms it connects. In a building near the door (or doors) is mostly located some information for visitors about a specific room, such as the name of the employee or name of the room. This way the building can be modelled for the application. The model of the map of the building is shown in Figure 12.

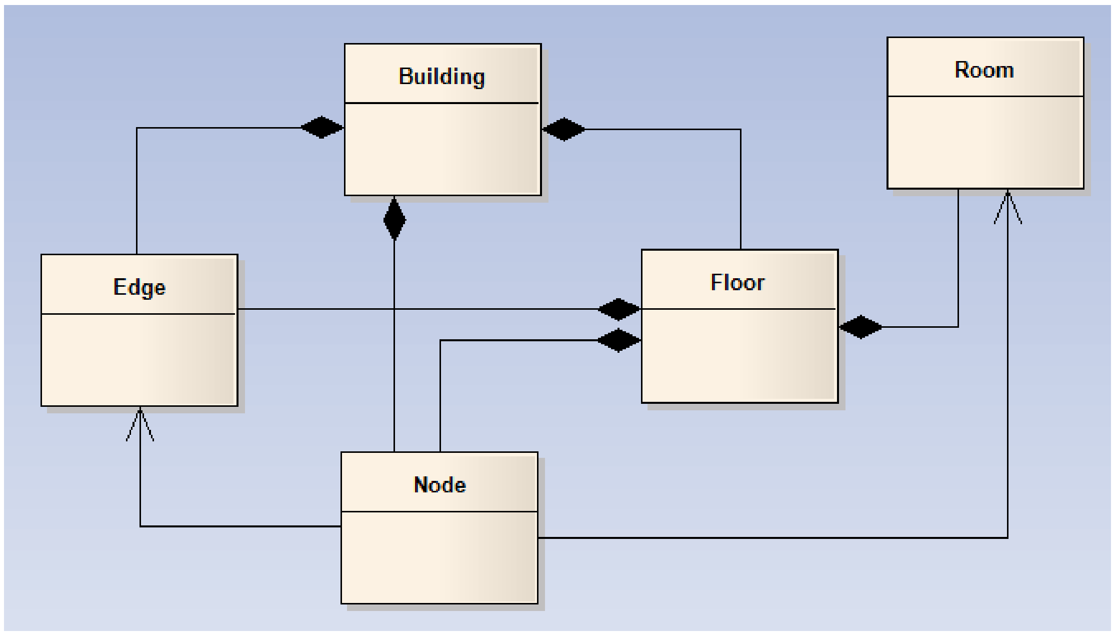


Figure 12. Diagram of building map model

The core class of the map model of the building is class Building. This class contains at least one instance of the class Floor which is a specific floor of the building. Another necessary element in the structure of the building is room represented by the class Room.

The doors were replaced by two separate classes Edge and Node in the implementation, which together with class Room can exactly identify a connection between two rooms. The main reason for implementing classes Node and Edge was using of these classes to find the shortest path in the building later. Node class represents the one **vertex** of the graph, which is used by Dijkstra's algorithm (Palúch 2008, 80). Subsequently, the class Edge is a representation of the one **edge** of the graph and thus it logically connects two instances of the class Node. Each instance of the class Node may have an instance of the class Room, in which the graph vertex is located. Lists of instances of classes Node and Edge are contained in classes Floor and Building. Floor class contains all the vertices and edges located on one floor. The parts of the class Building are lists of edges and vertices which connect the individual floors of the building. By lists traversal and determining which room belongs to vertex from the list (or vertex of Edge instance) it can be exactly identified where the vertex is located in the building.

Classes Building, Floor and Room are the main graphical representation of buildings (resp. parts of buildings). Edge and Node classes represent connection parts of the building; however, in addition, they are used for orientation in the map and also for finding the shortest path to the destination place in the building. Each instance of floors has also relative path to the background image file used for better user orientation on the map.

5.6 Application FriBasicMapEditor

The main function of this application is in the **modifying maps** for navigation, i.e. it shows the map modified right now. Another function needed in this application is the selection of a particular object on the map, displaying the information about this object and changing this information. Also, it is necessary to be able to assign two dimensional graphical codes to these objects. Because the application draws the map, it is necessary that this map has to be represented somehow by the application.

For the purposes of these requirements additional classes were created in addition to the main class *MainFrame*, namely:

- Class *MapCanvasForEdit* – it is responsible for rendering maps
- Class *RoomInfoPanelForEdit* – it displays information about selected object from the map
- Building Class – it represents the map of the building itself
- Class *QRWork* – ensures the generation and loading the QR codes
- Class *NewRoomDialog* – dialog window which creates a new room in the map.

These classes, along with the main application class, and with classes that represent the building can be seen in Figure 13.

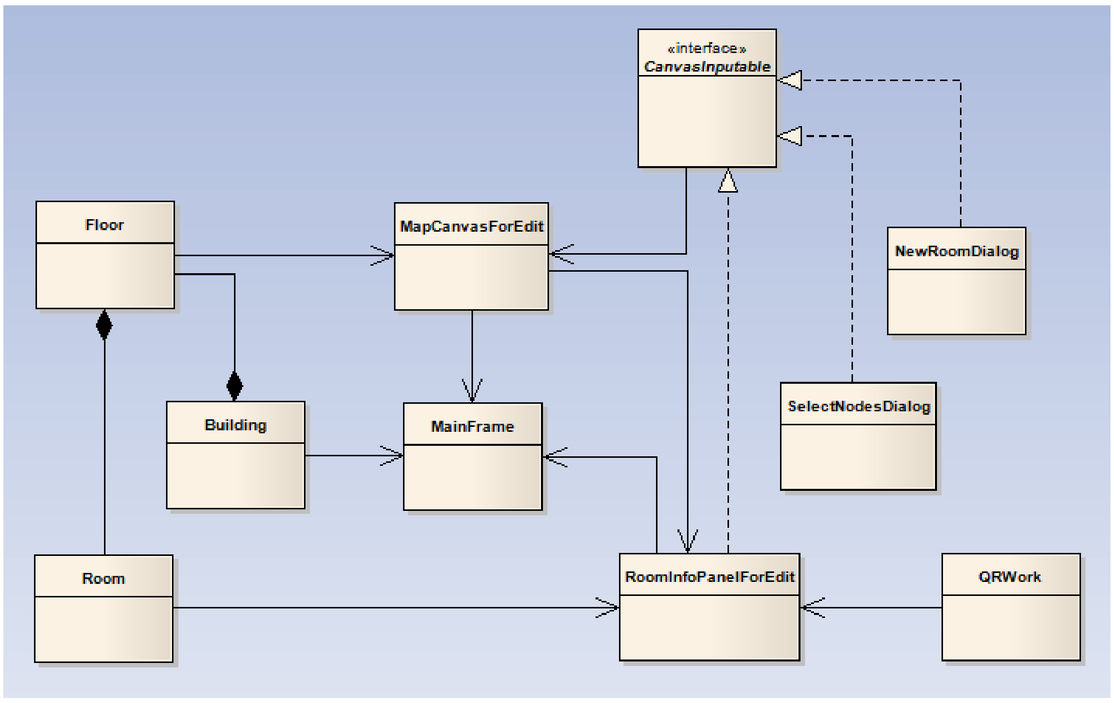


Figure 13. Main classes of *FriBasicMapEditor* application

The description of the most important classes is listed in next subchapters.

5.6.1 Class MapCanvasForEdit

Class provides the **rendering of the current map** and thus it must support loading of some maps. Because it shows only one floor of the building, it is necessary to be able to load, edit and save only the one floor. It must also be able to handle user actions, i.e. analyse and evaluate clicking action on a graphical representation of an object displayed on the map. For more comfortable usage of displayed map, it is good to implement tools for zooming and moving the map inside the class.

Summary of the properties of the class *MapCanvasForEdit*:

- Maps rendering
- Maps loading (floor loading)
- Zoom in / out
- Map dragging
- Actions recording - selection (or moving) and evaluate them.

MapCanvasForEdit class is implemented as a **subclass of *JPanel***, so it provides basic functions from parent class and also it provides the functions required for the assignment.

The main part of the class *MapCanvasForEdit* is the map background *MapBg*. The actions are recorded and performed only above this background.

Map drawing ensures the overwritten method *paint()*, which extends the basic inherited *paint()* method by rendering all the rooms created over map. The loading of specific floor of map provides method *load(floor)* with a parameter as a reference to object floor. This reference will be saved in class attribute for later work with this floor (floor editing).

The zoom functionality is implemented by methods *zoomIn()*, *zoomOut()*, *setToOriginalZoom()* and *performZoom()*. The first three mentioned methods only set the zoom value. Method *performZoom()*, which is a part of each of these methods, applies the set zoom settings and redraws the entire map.

The map dragging is also related to events capture. If this feature is currently enabled, the point where the mouse button was pressed is recorded into attributes *m_XDifference*, *m_YDifference* and after this "pulling the click" action the entire map is redrawn on new coordinates every time the user moves the mouse. This function is only possible if the current size of the map is larger than the used content area of *MapCanvasForEdit* instance.

5.6.2 Class RoomInfoPanelForEdit

This class displays **information about the selected object** on the map. Rooms are objects primary accessible for the selection. This means that the main information displayed will relate to specific room. Thus, it is needed to view the **name** and room **number**, some additional information assigned to rooms and possibly a direct connection to the other rooms. The class needs to have a reference to the room for reliable access to room information.

CanvasInputable interface is already implemented in this class and therefore it accepts the receiving messages *acceptRoomInfo(room)*. The class can receive and work with any reference to an object with type Room by this implemented method.

Because of the fact that a class has access to all room features through reference, it is good to take this opportunity also for changing the data in room. Thus the class implements methods for updating the room information: *jButtonEditNameActionPerformed(evt)* and *jButtonEditNumberActionPerformed(evt)*.

Another of the implemented functionalities permits to **generate** and to **print QR code** which represents graphical reference outside applications. This is implemented in methods *generateQR()* and *printQR()*.

The class is ready implementing additional functions for editing another properties of rooms, for example schedule of occupancy for this room, list of teachers (employees), who used this room etc.

Thus, the instance of class *RoomInfoPanelForEdit* is also able to display room breakpoints of this room. The room breakpoints are shown as a list of points.

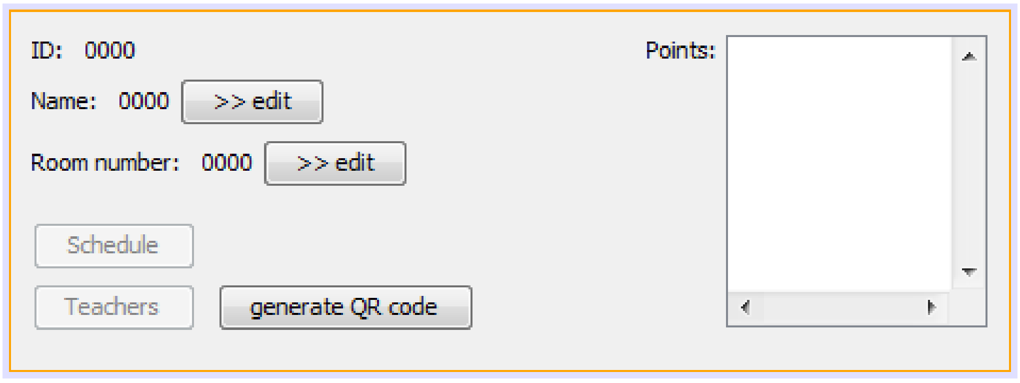


Figure 14. Form window - class *RoomInfoPanelForEdit*

5.6.3 Class NewRoomDialog

This class is responsible for **adding** a new **room** to the map. Each room added to the map has to have a unique identifier. Better clarity for the user is to call room by name, also it is good to add a description and set the room number according to the structure of the building.

Since the map can contain **three types of rooms**, this class must also implement this option. It can be chosen whether a user adds an ordinary room or stairway (or elevator) where “the only one per building” rule is not applied.

The class is implemented as a subclass of class *JDialog*. Inherited method *setVisible(true)* causes that the instance is visualized. For creating new room into the map it is needed to enter a room name and room number. This number must be a numeric type, if not, the application will notify this by a special dialog window. Further, the room has to have assigned at least three points on the map, because it is necessary to represent the room somehow graphically.

For adding a special room like stairs or elevator, the specific type by the objects of type *RadioButton* can be chosen. If the user wants to add a stairway or an elevator which already exists on another floor, i.e. connect two floors, there is a list loaded with all previously added stairs or elevators. It is necessary to store a reference to the whole map building for updating these lists. The reference is stored in the attribute called map the type of which is Building. Then the data is obtained from this reference for lists of type *JList*.

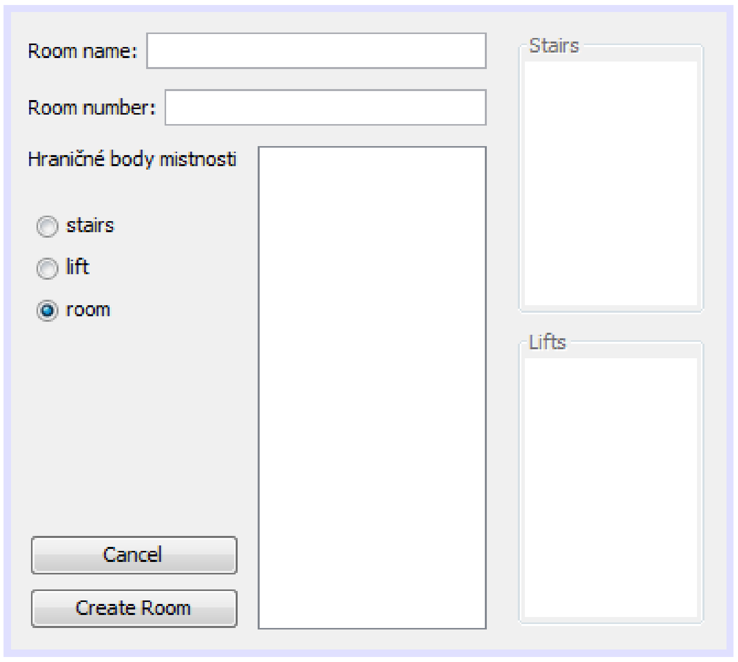


Figure 15. Form window - class *NewRoomDialog*

The adding itself is protected by setting the sender attribute of type *MapCanvasForEdit* which **implements** the method for **adding** a **concrete room** in the currently loaded map. This attribute stores a reference to the object, where the current map is loaded. After pressing the button for adding room it checks all data fields, creates a new object (or an existing selected object is loaded) and sends the message *addRoomToMap* reference (room) through the sender with the parameters of the specific room.

5.6.4 Class Mainframe

This class is the main class of the application which provides access to other elements. The class has access to the loaded map (Figure 16). This map has to be loaded from object file at first. It is also necessary to save this map as a file object after modifying. It is therefore necessary to implement some management for loading and saving the current map.

Next, it must provide communication between *MapCanvasForEdit* class, which ensures the actions recording on the map and class *RoomInfoPanelForEdit* for extracting information about the selected room.

It is necessary to provide the user a possibility to select the current floor, whereas *MapCavas* shows only one floor. Besides selection also the adding and removing floor function is required.

The major activity which the class has to be able to ensure is the **updating** of currently loaded **map**. It must be able to add a new room and add a new door for what it uses classes *NewRoomDialog* and *NewDoorDialog*. It is also, however, necessary to provide changing of properties of the currently selected room or removing the room from the map.

Class *MainFrame* is implemented as a subclass of *JFrame* class from JDK. It represents the main application window and provides the aforementioned **connections** between individual **parts of the application** such as floor map displaying, showing the information about the selected room and other mentioned in the text above.

The main activity is mediation of the functions of *MapCanvasForEdit* class to other classes. An instance of this class is as a window of user interface which includes a floor map. This map is an instance of *MapCanvasForEdit* class. It is stored in the attribute *mapCanvas*. The events registered by instance *mapCanvas* are forwarded to the instance of *RoomInfoPanelForEdit* class, which also is accessible via the attribute *roomInfoPanel*. Class Mainframe provides an initial setup of communication between these two instances. When the instance *roomInfoPanel* is created it is the reference to *mapCanvas* which is also set through the calling method *setSender(mapCanvas)*. Because the reference to the object in attribute *mapCanvas* is changed when the current floor is changed, the message for setting this communication is called every time when the new map is loaded, even if it is an empty map. This communication is set by calling method *mapCanvas.setOutForRoomInfo(roomInfoPanel)*. This provides collaboration in the selection of objects on the map and changing them by using these instances.

A similar connection is created for showing a list of created rooms on floor. The list is filled by calling method *updateRoomList()*. The reference for the designation of a specific room in the list which was selected, is set by calling the method *mapCanvas.setRoomList(jListRoomList).* On the other way the room indication on the map is provided by catching the selection event in room list *jListRoomList*.

The loaded floor changing is implemented by using instance *jListFloorList*. After the events of the selection are caught on this component an object is selected on the map, respectively floor is loaded into the instance *mapCanvas*.

Furthermore, the class *MainFrame* implements buttons for dialogs creating, with which the new room or door can be added to the map. Adding a room is implemented using the method *addRoom()*, and the method *addDoor()* provides adding doors into the map.

Implementation also includes a user interface for zooming methods invocation from instance *mapCanvas*. These methods are *zoomIn()*, *zoomOut()*, *setToOriginalZoom()* which are included in methods *jButtonToolbarZoomInActionPerformed(evt), jButtonToolbarZoomOutActionPerformed(evt), jButtonToolbarOriginalZoomActionPerformed()*.

Each map of building can be saved and reloaded back. When saving the whole created map the object is stored in one object file by using instance of *FileOutputStream* class. File path is specified by the user through saving dialogue. The whole saving process is contained in the *saveButton()* method. Opening of already created map works similarly like a saving. It is called by the method *openButton()*. Opening principle is based on setting the basic values of attributes of *MainFrame* class, this occurs by calling *setNewMapCanvas()* method. After the file for map loading is selected, the map is loaded into variable map through the instance of the class *FileInputStream*, one floor of this map is loaded, the rooms list and a list of floors is updated, and the communication between *mapCanvas* and other parts of the program is also set. Following this, the area which displays the map is updated.

The creating of the **new map** is very similar than the opening of an already created map. It creates a new instance of the class Building, which is stored in the map attribute. Lists of rooms and floors are restored, and references for communication with *mapCanvas* instance are set. Subsequently, the area for map displaying is redrawn and thus restored.

The main window can be seen in Figure 16.

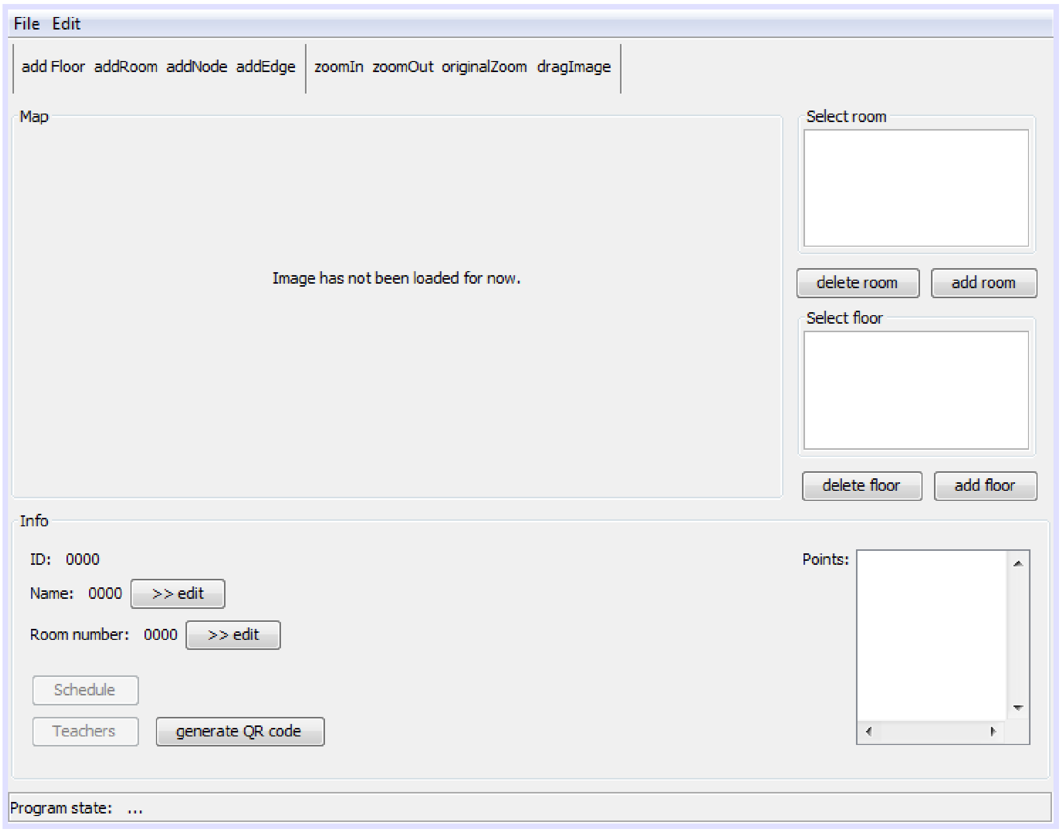


Image has not been loaded for now.

ID: 0000

Name: 0000

Room number: 0000

Program state: ...

Figure 16. FriBasicMapEditor application - main window

5.6.5 Class QRWork

The tools for **generating** the **QR codes** should be implemented for assigning QR codes to the individual room. Each room should be represented as the unique object in order to avoid miscalculation when navigating. Each room contains an attribute id which is uniquely assigned to only one object in the map. This attribute, since it clearly represents a room on the map, will be used for representing room in QR code.

This class thus implements methods for generating QR codes. Specifically, the static methods:

- *generateMatrix (room)* - provides creation of QR code bitmap matrix from the desired room. It is possible to create an image from this matrix which can be printed later.
- *generateQRImage (matrix)* - creates the image of type *BufferedImage* from the desired bit matrix
- *saveImageToPngFile (image, filePath)* - saves the specified image to the file specified by file path

Information about the room is inserted into a QR code in format similar to XML. This format was chosen for the good quality of word processing, as well as being quite understandable and readable for the user. The readability is an advantage if the user is not using this navigation system. After decoding information from a QR code it is possible to read in front of which room the user is currently located, and thus the current location.

Example of embedded information in QR code about the room:

<room_name> agency </ room_name>
<room_number> 325 </ room_number>
<id> 2 </ id>

5.7 Application FriNavigate

The main focus of this work is **navigation in building** using QR codes. It is possible to find the shortest path to the selected room on the map created by FriBasicMapEditor application. It is necessary to interpret this map to the user correctly and also to the navigation application. Navigation itself would not dispense without the function for entering the **destination** position and the **current position** where the user is currently located.

For updating the position of the user the application needs to know to decode QR code printed and placed next to a specific room. QR code is needed to load the application before the actual decoding. When navigating in the building, an image of the QR code is printed and placed to the wall or other visible place near the specific room or object.

There are two options offered for loading the QR code. The first one is to use application interface for communicating with a camera or video camera and dynamically to scan the printed image. For this option the Java Media Framework could be used. The second option is that digitizing a printed image is exempted from the application and the external tools for QR code image scanning is used which scan the QR code and store it to the disk, from where it is downloaded by application.

Because of time opportunities the second way of working with stored images was selected, thus the digitizing method of image depends on user. The external library ZXing, mentioned above, was used for decoding loaded image. After loading the current location and destination place, it is necessary to **calculate** and thus to find **the shortest path** to the entered destination.

The implementation is mostly based on the classes already used in the application *FriBasicMapEditor*. It could be said that *FriNavigate* is the impoverished sister of the previous application. The applications have the same basic structure, however, they differ in some structural elements.

FriNavigate is used for displaying map *MapCanvasForNavigate* class which extends *MapCanvasForEdit* by the possibility of communication with the dialog for room picking. In addition to this extension, the class is identical with class *MapCanvasForEdit*.

The aforementioned depletion lies in the **removal of features for** maps **editing**, such as **adding** and **removing rooms**, **floors**, **vertices** and **edges** of the graph, also map **saving**. Functions for zoom in, zoom out and map moving stay available still. Loading of generated maps is also accessible.

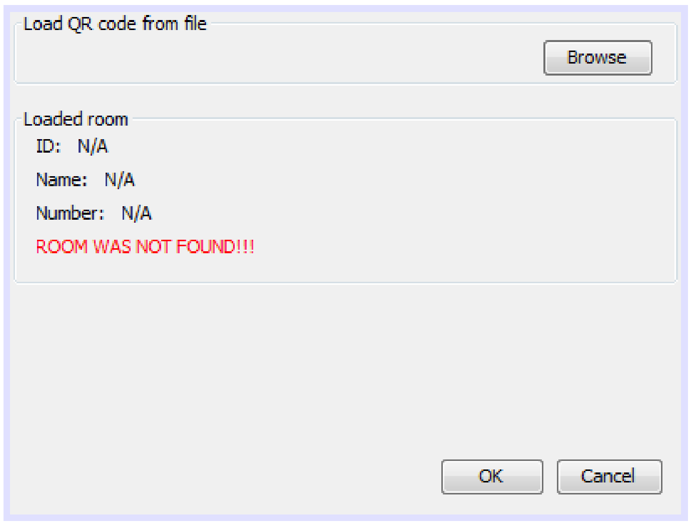
The class for displaying information about the selected room was modified similarly. Class *RoomInfoPanelForNavigate* displays only information about the room, it does **not provide the ability for changing** the room data. The **new features** which cannot be found in the application *FriBasicMapEditor* include **updating the position** of a user, **selection** of the **target** room and **calculating** resultant **path** to the destination room.

The functionality is implemented in the two new classes, namely:

- *LoadLocationDialog* – Class updates the user position
- *SelectRoomDialog* – Ensures the selection and transferring the target room reference to main class.

5.7.1 Class LoadLocationDialog

Class provides **QR code loading** and **processing** feature for user. An instance of this class as a subclass of *JDialog* class is called from the main class *MainFrameNavigate* and the main class object is given to this instance as a reference. Through this reference the *LoadLocationDialog* has access to the currently loaded map.


Load QR code from file

Browse

Loaded room

ID: N/A

Name: N/A

Number: N/A

ROOM WAS NOT FOUND!!!

OK

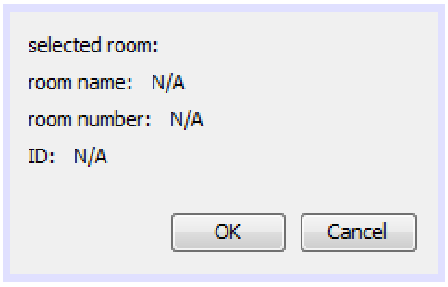
Cancel

Figure 17. Dialog window – class *LoadLocationDialog*

Bearing in mind that it will use QR code images already created, this class provides a possibility to choose a particular file with cooperation with instance of *JFileChooser* class. It can choose one of the following types of graphics files: files with the extension .jpg, .jpeg, .png, or .bmp. After image selection, the image is automatically processed by the method *decodeQR()* of class *QRWork*. The information needed for room identification encoded in QR code is collected and processed by using an instance of *QRTextParser* class, by method *parseByLine()*. These data are compared in *findRoom()* method with existing rooms in map. When the match is found the reference to the found room is stored and the information about the found room is written on a dialog window. After confirming the dialog the attribute of the current position in the main class of the application is set using selected room reference and reference to an instance of the main window (attribute of current location - *location*).

5.7.2 Class SelectRoomDialog

For the **selection of the destination** of path this class is used. Like the previous class, an instance of this class is called from the main application class *MainFrameNavigate*. The class is implemented as a subclass of *JDialog* class and it implements an interface for communicating with *CanvasInputable* area, where the map of the building is drawn.



selected room:

room name: N/A

room number: N/A

ID: N/A

Figure 18. Dialog window - *SelectRoomDialog*

After an instance of this class is created the instance of the class *MapCanvasForNavigate* is informed that it should send information about the click event on some graphical representation of the room to the entered instance of *SelectRoomDialog* dialog. After the information about the selected room is stored this dialog shows this information to the user.

The dialog will send a message for **setting the destination room** to the instance of *MainFrameNavigate* class after dialog confirming.

5.7.3 Class MainFrameNavigate

As already mentioned above, the application *FriNavigate* is an **impoverished version** of *FriBasicMapEditor*. The styling and construction of the application is based on the application *FriBasicMapEditor* that the main application window reflects. Class *MainFrameNavigate* contains the main window.

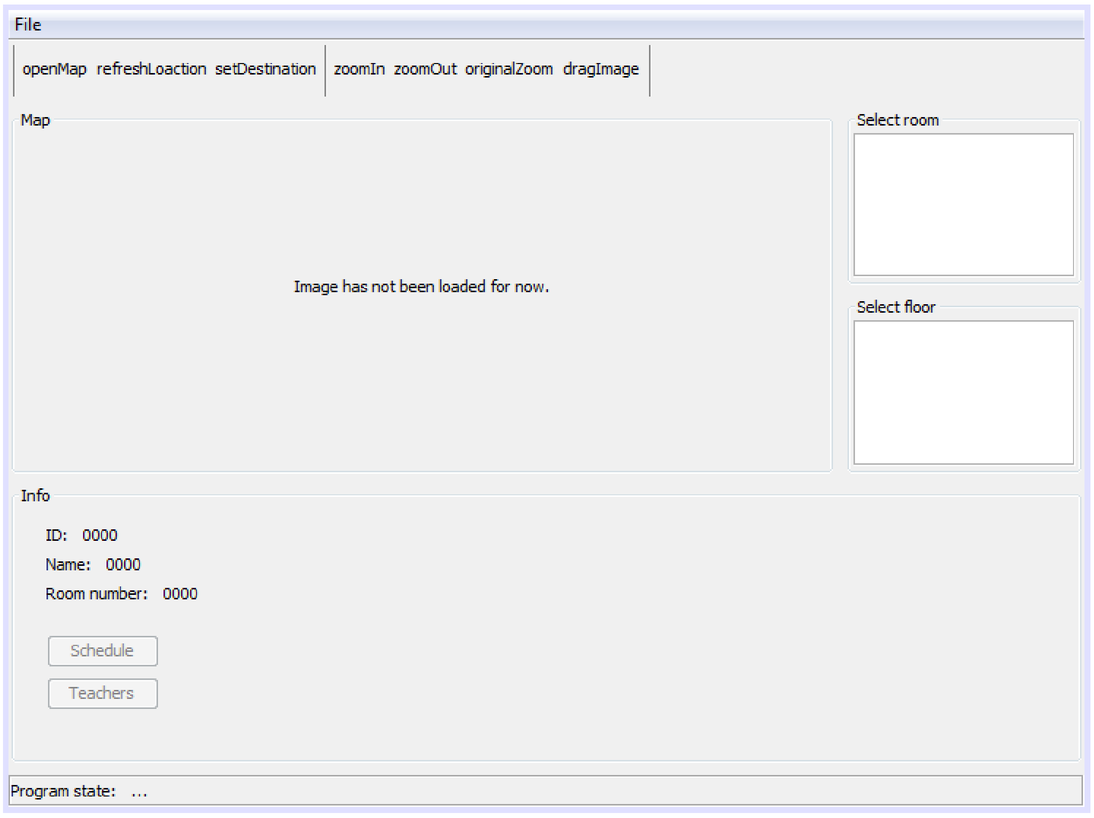


Image has not been loaded for now.

ID: 0000
Name: 0000
Room number: 0000

Program state: ...

Figure 19. Main window of *FriNavigate* application

Class provides access to the main functions called by user. It implements the same approach to functions zoom in, zoom out and map moving. In addition, it provides features for **setting the current** user **position** in the building and **setting the destination room** where the user wants to get through this navigation. These options are provided by methods *refreshLocation()* and *refreshDestination()* which create a relevant dialog. Subsequently dialogs send messages to this instance by methods *setDestination()* and *setLocationOnMap()*, which set necessary references to selected objects.

After the user has selected the current position and the destination room, the method *findAndShowPathToDestination()* is automatically called. This method **searches the shortest path** from the entered current position to the target room and after the path is found it is displayed on the map using the method *showPath()*. The shortest path is found by searching a graph using an instance of class Dijkstra.

5.7.4 Class Dijkstra

This class provides **calculating of the shortest path** from one point to all other points of the specified graph. Class, as the name suggests, is an implementation of **Dijkstra's**

algorithm for finding shortest paths, which is described in Chapter 2. (Palúch 2008, 80)

The constructor of this class contains lists of vertices and edges of the graph. These objects are later needed to calculate the shortest path, and these lists contain two-part records. The first element of the list item is the ID number of the graph node, resp. ID of edge. The second element is a reference to the node itself, resp. to edge.

The calculation itself, respectively finding the shortest path runs inside the method *dijkstrovAlgoritmus(idStartNode)* where the parameter is an identifier of the initial node, where the path will start. The method finds the shortest path from the specified node based on searching the lists of edges and nodes and writes the **result** to the resultant **list** where the list item is composed of 2 elements, resp. 3 elements. Each record is represented by the ID number of the peak and two-dimensional element (therefore generally the record **contains 3 elements**) contains the length of the found path to this node and the ID of previous path node in graph.

The record in result list has the following form:

Table 4. Format of single record in result of Dijkstra's algorithm

Node ID	Path length
	Previous node ID

The detailed documentation of all classes is in the attached *JavaDoc* documentation. (Appendix 1)

6 DESCRIPTION OF CREATED APPLICATIONS

A map of the building, where the indoor navigation is applied is needed and where the application itself will be used. This map can be created using *FriBasicMapEditor* application.

6.1 FriBasicMapEditor

The application is composed of some graphical, functionally distinct parts. The Figure below describes the single elements which can be found in main window of *FriBasicMapEditor* application.

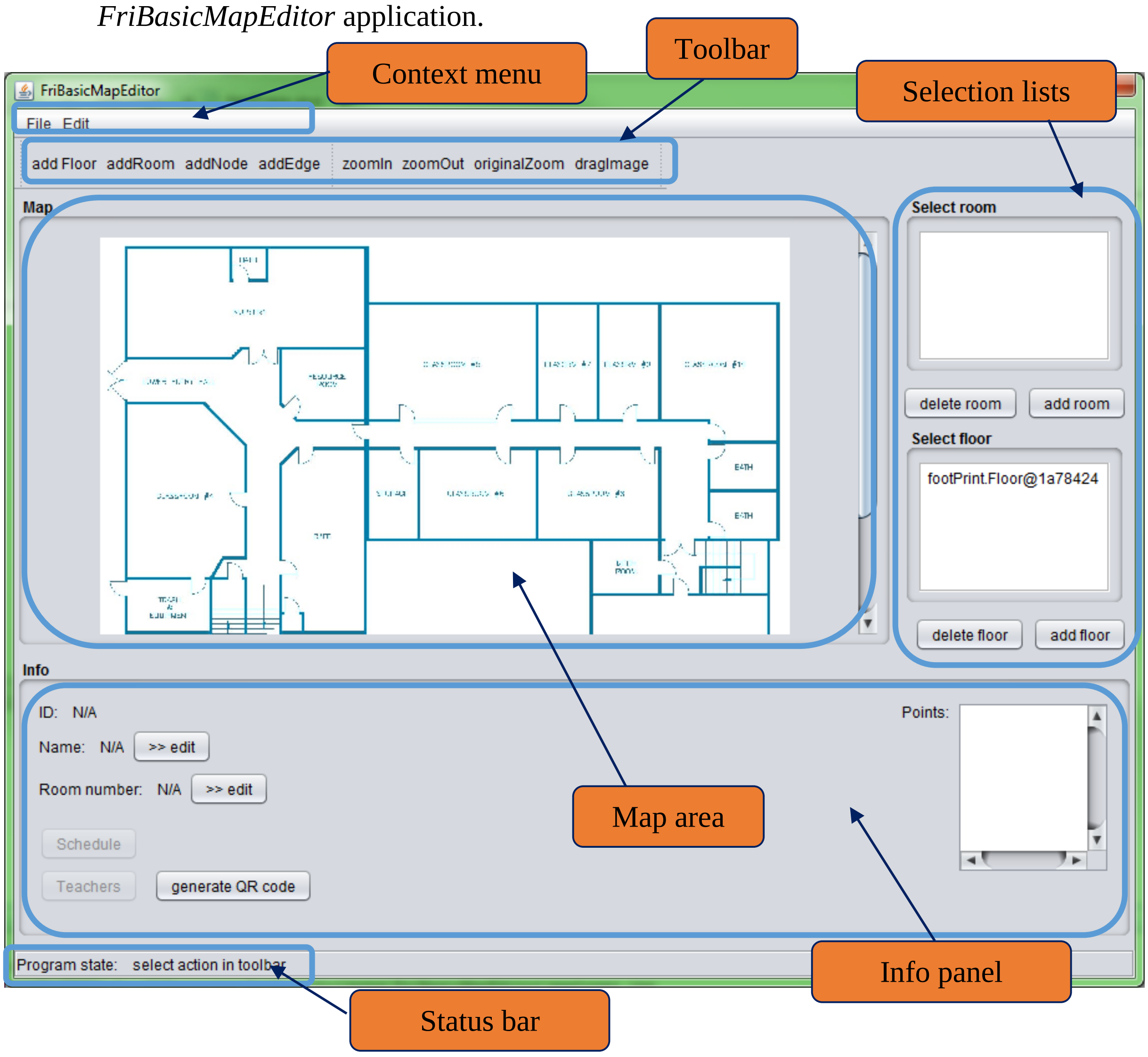


Figure 20. Elements of *FriBasicMapEditor* application

- **Main Menu** – Using the main menu it is possible to access the basic functions. It contains two items *File* and *Edit*.

- After opening the main menu the *File* menu item is displayed. This menu provides buttons for creating a new map, opening a map already created, saving and also a button for exiting the application.

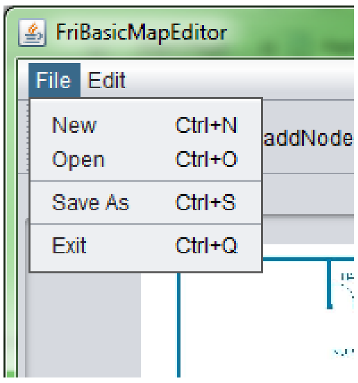


Figure 21. Overview of the context menu *File*

- Menu item *Edit* includes options for loading map background and options for working with the map view. It provides possibilities for adding a new floor, room, node of the graph and the edges between nodes.

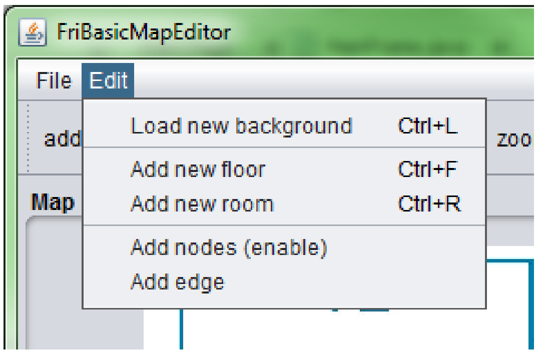


Figure 22. Overview of the context menu *Edit*

- **Toolbar** – It contains all options for map editing. These options are also in the main menu under *Edit* item. In addition to these editing tools, it contains also options for manipulating with the map, such as zoom in, zoom out and moving the map.
- **Map** – This part is the most important part of the application because it shows the map created by user and provides the interaction between the user and the map model.
- **Selection lists** - are designed for a selection of rooms and floors created on the map.

- **Info panel** – It shows the information on the selected object on the map and provides functionality to change this informations, also it can generate a QR code for the selected map object.
- **Status bar** – It shows a text message for users which explains the current state of the application.

6.1.1 Map creating

The basis for map creating is loading the background of the map. It is possible to create an own map of the building over this background. Loading the background of the map is invoked by context menu *Edit - > Load new background* or by pressing keys combination CTRL + L.

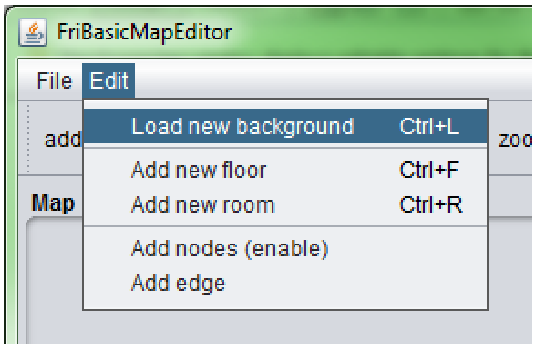


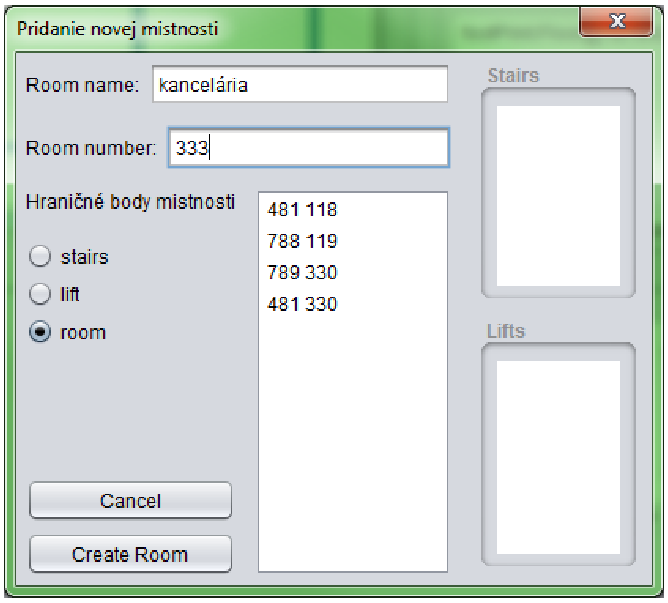
Figure 23. Loading of map background

After the file with background is selected and dialog window is confirmed, the image is loaded into the application. The basic element of map is ready.

For **adding** a new **room** it is needed to select the appropriate item (*Add new room*) from the main menu, from toolbar or click on the button under the room selection list.

Dialog window for creating a new room which is opened after previous action requires to enter the **room name** and **room number**. Also, it is necessary to select the room on the map by clicking on the map. The point where the user clicked is transferred to the list of border points in the dialog and stored as a border point of a graphical representation of the room. It can be chosen from three types of rooms for determining the **type of room**, namely ordinary room (room), elevator (lift) and stairs. The dialog also contains two selection lists for created elevators and stairs, which become available in the case of choice of appropriate type of room. From these lists, user can also select an existing room in case user wants to con**nect two different floors**. This selected room will be copied into the just loaded floor, thereby the link between floors are achieved (since the room will be added on two floors in one map model).

Figure 24 describes the dialog window for adding new room to the map with entered name, room number, selected room type and selected 4 border points.

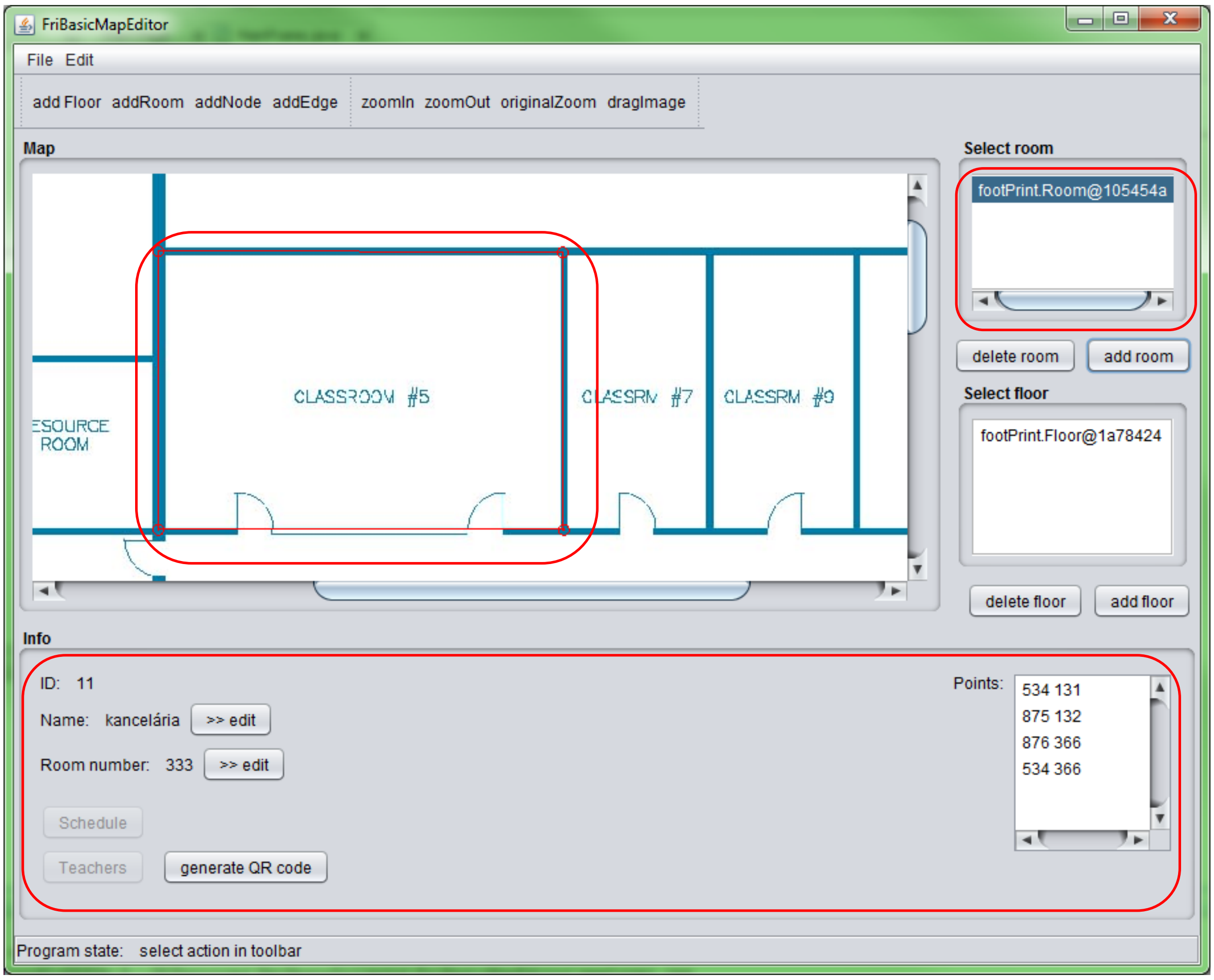


481 118
788 119
789 330
481 330

Figure 24. Dialog window for creation of new room

After dialog confirmation the room is created and added to the map. The selection list of rooms is also updated. The application now allows to select the created room and shows information about this room in the Info panel section. This way it is possible to add all the rooms to the map.

Figure 25 shows how the main window are changed after the new room is added, when the dialog for adding new room is confirmed.



footPrint.Room@105454a

footPrint.Floor@1a78424

ID: 11

Name: kanceléria

Room number: 333

Points:

```
534 131
875 132
876 366
534 366
```

Program state: select action in toolbar

Figure 25. Updated application after room creation

Connections still need to be created between the rooms after they have been created. These connections are formed by vertices and edges. After activating the *Add node* option it will be possible to add the vertices to the map without restrictions. It is expected that the **vertices** are added to the **locations where** the **user should decide** how to continue to destination place. When the vertices are added it is needed to connect these nodes and thus to create a graph of the building, which is possible by creating an edge between two created nodes.

After selecting *Add new edge* the dialog for selecting two vertices is opened. When the node vertex is selected in nodes list this is also highlighted on the map. For creating an edge between vertices exactly **two vertices** must be selected. After the selection and dialog confirmation the edge is created and drawn on the map.

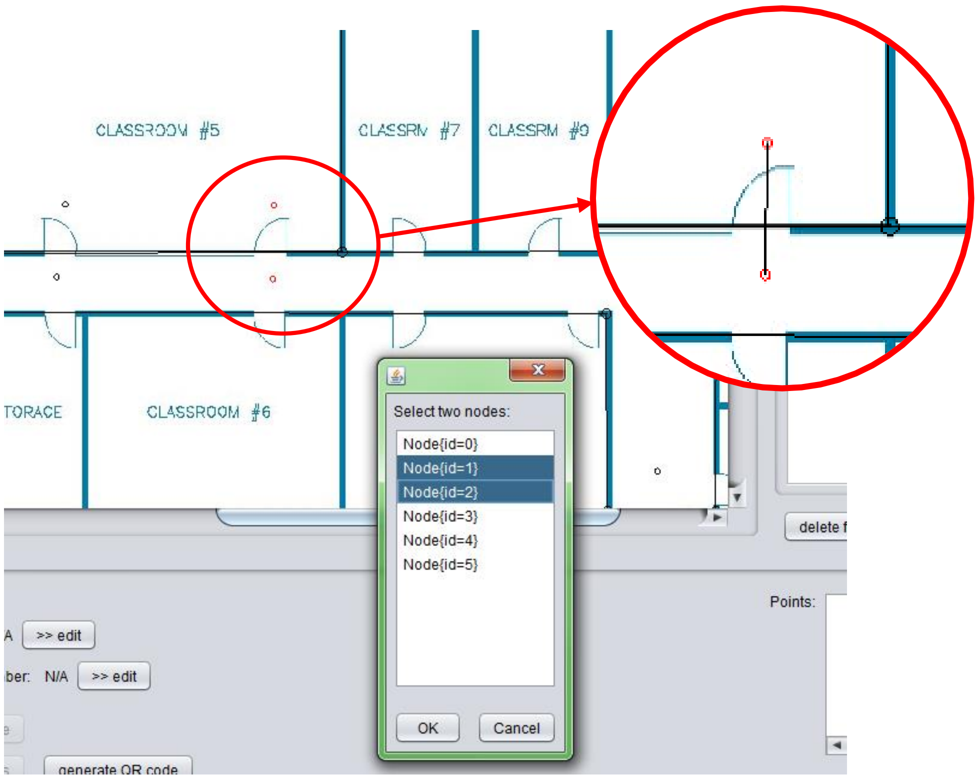


Figure 26. Changing the picture of map after edge is created between two vertices (points)

Thus, it is necessary to set all the connections between vertices, including the connections inside the rooms. The vertices themselves may not only be used for connecting the rooms, but the room can be divided into a number of places where the user can change the direction. When a map is being created it cannot forget to create an edge (via the *Add edge* option) between all the directly connected adjacent points.

The node which is not located in the room is considered to be a node **outside the building**, and therefore after this node is connected to some node in the building, the created edge is regarded as the **entrance** to the building.

6.1.2 Rooms editing

Following Figure 27 where the selected room is displayed in the Info panel, editing this room is the next step. After clicking on the *edit* option next to the room name or next to room number these values can be **changed**. User has to enter a new name of the room to the opened dialog, resp. room number and confirm the dialog for saving the changes of value.

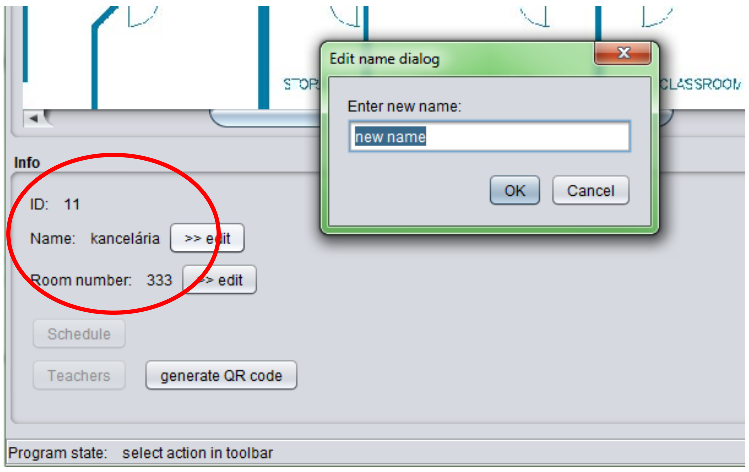


Figure 27. Calling the dialog for changing the name of room

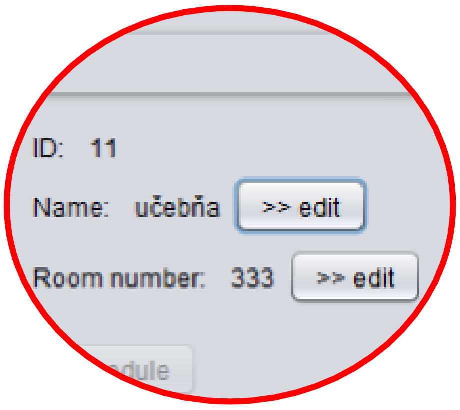


Figure 28. Changed name of the room

It is necessary to **generate** appropriate **QR code** for every room for navigation. By selecting the option *generate QR code* in Info panel it opens a dialog for selecting the output file where QR code will be stored and from where user can then print the code.

6.2 FriNavigate

When the map is created and QR codes are generated it is possible to use this map for navigation. For this the second application, called *FriNavigate* is designed, the style of which is similar to application *FriBasicMapEditor*.

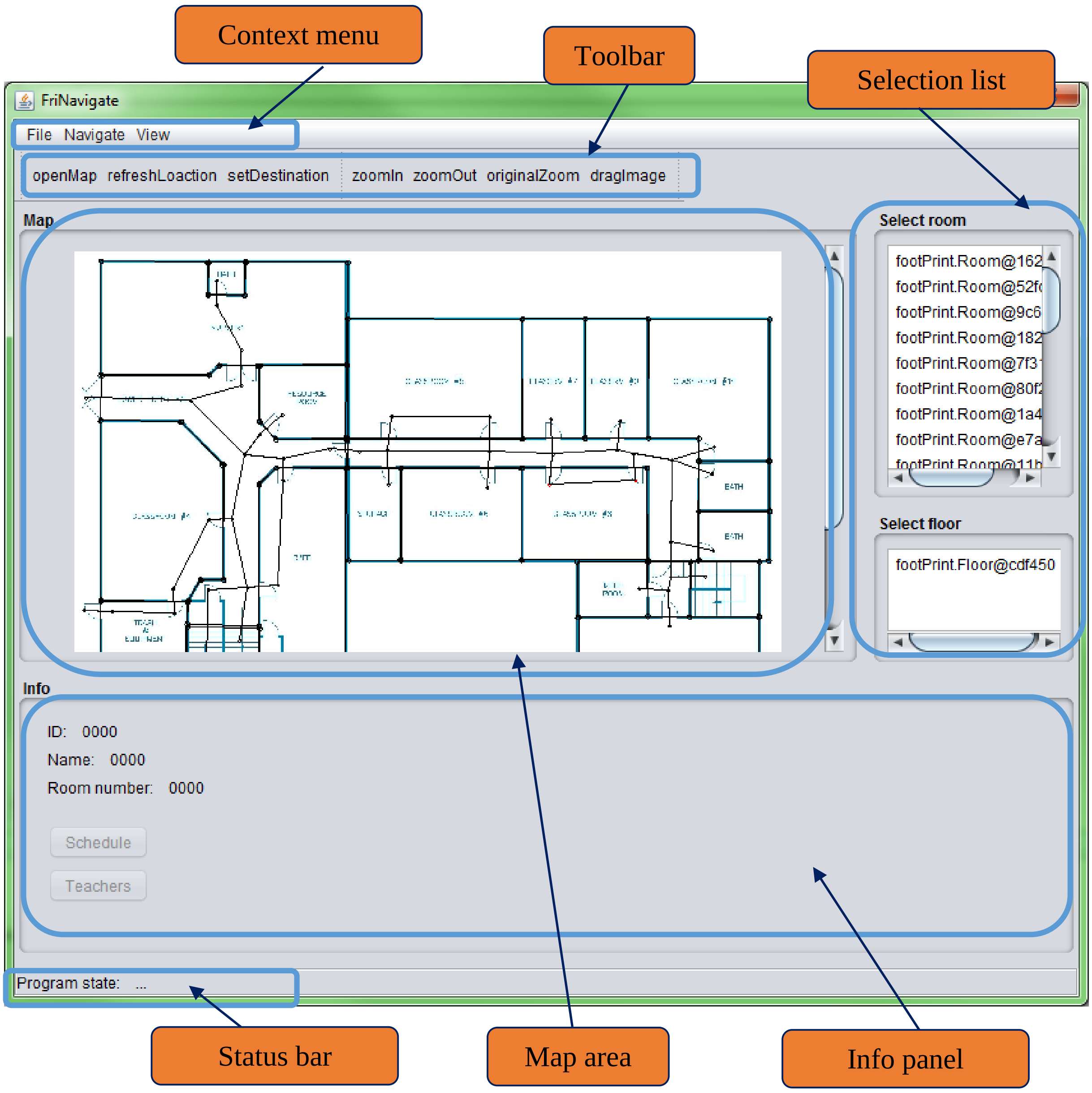


Figure 29. Elements of application *FriNavigate*

Application window also contains the menu bar, toolbar, area for the map displaying, rooms and floors selection lists, Info panel and status bar.

Application *FriNavigate* provides for user other features compared with application *FriBasicMapEditor*. The difference is noticeable already in the main menu, which contains the following elements:

- *File* – It provides option for opening maps of the buildings and option for exiting the application.

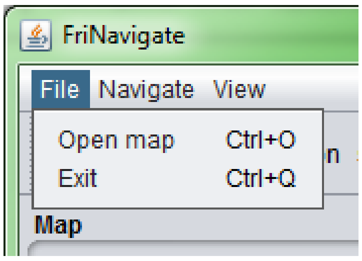


Figure 30. Overview of the *File* context menu in application *FriNavigate*

- *Navigate* – Options included in this menu are the basic options for **setting** the user **path**.

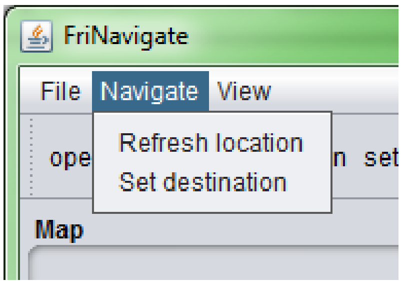


Figure 31. Overview of the *Navigate* menu in the application *FriNavigate*

- *View* – This menu contains options for **working with** a displayed **map** as zoom in, zoom out, turning on and off the map moving by dragging the mouse.

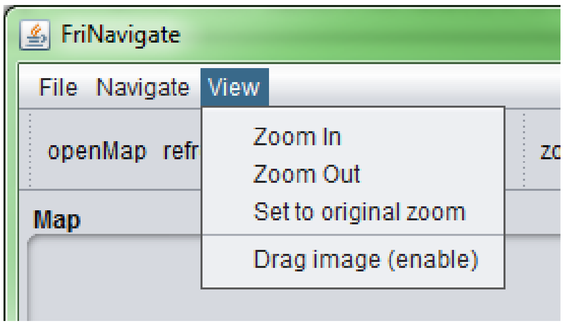


Figure 32. Overview of the *View* menu in the application *FriNavigate*

The element *toolbar* provides **quick access** to menu options. This means that it contains almost all elements contained in the main menu.

The part of the *Info panel* is depleted room editing functions, however, like in *FriBasicMapEditor* application it displays the **information about** the selected **room**.

After the map is loaded by clicking on the option *Open map* in context menu, the application is ready to navigate the user to any available place in the building.

6.2.1 User position actualization

The core part of navigation is determining from where and to where the user should be navigated. The *Refresh location* option in context menu *Navigate* is designed **for setting** the place where the navigation should **start** (only if the map has been loaded). Upon activation of this option dialog Load location is opened. The dialog contains options for selecting the image of generated QR code of particular room. After QR code on the image is recognized, the information about the loaded room is displayed in *loaded room* area of the dialog. The current position is set after the dialog is confirmed by button OK.

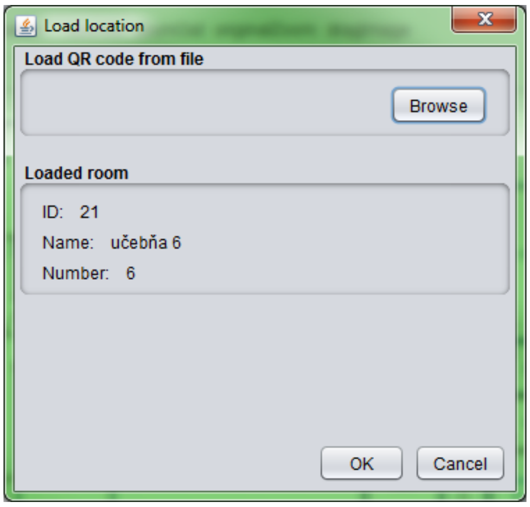


Figure 33. Dialog for updating the user's location

After the current position of the user is set, this information is displayed in a map via **highlighting** the **room** where the user is located.

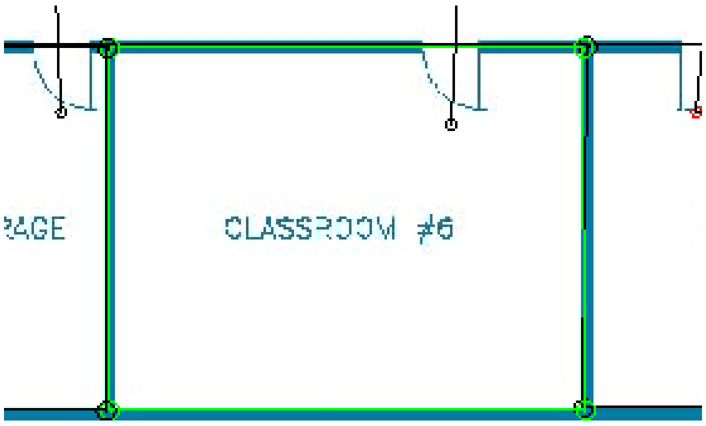
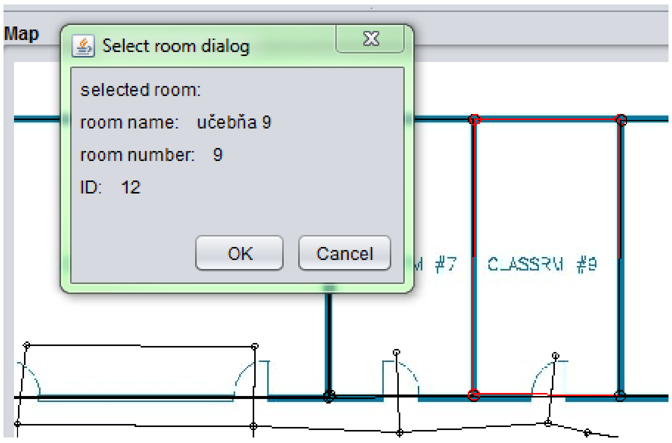


Figure 34. User's current position displayed on the map

6.2.2 Set destination of the user path

The option *Set destination* is designed for **determining** the **destination** of the path. After this option is selected, the dialog which contains information about the currently selected room on the map is displayed. If the user selects any other room from the map the information in the dialog is refreshed to the currently selected room.



selected room:
room name: učebňa 9
room number: 9
ID: 12

Figure 35. Setting the destination room

After confirmation of the dialog the destination room is set and navigating starts the path calculation.

The **path calculation** is performed with any other **change** of the current user **position** or **destination** room.

The **found path** is then highlighted on the map by red colour.

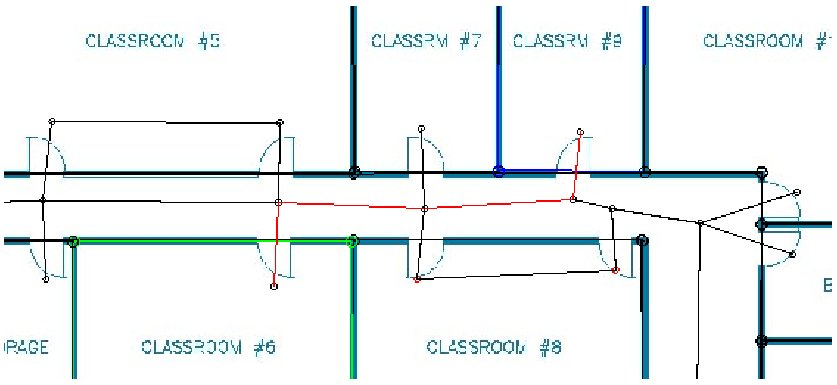


Figure 36. Path found from the current user position to the target room is displayed

7 DISCUSSION

The aim of this research project was to create an application which is able to locate the current user's location based on the recognition of two-dimensional codes. It is able to navigate to given place using this application. According to the assignment, the application should also ensure creation of two-dimensional maps and to assign additional information about objects to the map, and thus to provide functions for generating and loading two-dimensional graphic codes.

All of these set objectives have been met. The recognition of printed QR codes using dynamic image processing cooperated with camera has not been implemented yet.

Based on the requirements I solved the study by dividing work into two parts by implementing two applications. The first part was about developing the application for creating and editing two-dimensional maps of the building, and thus the application *FriBasicMapEditor*. The second part of the work is the application *FriNavigate* as the application for navigating the user from one place to another in the building using QR codes for localization. The first application provides generation of QR codes and assigning them to the map, the second one recognizes the generated QR Code and on the basis of this recognition identifies the room on the map. After the destination place is selected by user the shortest path to this target place is displayed on the map.

Each of these two applications is possible to improve despite the fact that the requirements were met. *FriBasicMapEditor* application is ready for implementing the options for adding other additional information of the object on the map, such as adding occupancy schedule of classrooms or functions for determining who owns the this room as an office. The map making could also be further improved in the application, resp. map displaying, to add multiple supported formats of maps commonly used for buildings maps and thus creating the advanced map editor from application *FriBasicMapEditor* for different formats. Similarly, the application *FriNavigate* could be supplemented by components providing localization using other technologies. I see a continuation of this work further in the creation of applications for mobile devices such as smartphones and tablets, through which the designed and developed applications would be practically usable.

REFERENCES

Android Developers Portal. N.d. *Page on developer.android.com.* Accessed on 22 January 2012. Retrieved from http://developer.android.com/

Ausmeier, B. 2011. *Indoor Navigation Using Mobile Phones*. [*Honours project report*] Accessed on 1 March 2012 Retrieved from http://people.cs.uct.ac.za/~tcampbell/project/resources/brett_report.pdf

Babinec, A., & Vitko, A. 2010. *Histogramové navigačné algoritmy - vývoj a princíp.* [*Journal article AUTOMA*]. Accessed on 1 April 2012 Retrieved from http://www.odbornecasopisy.cz/res/pdf/41059.pdf

Demaine, E. 2011. *6.006: Introduction to Algorithms.*Accessed on 5 March 2012. Retrieved from http://courses.csail.mit.edu/6.006/spring11/rec/rec15.pdf

Denso Wave Incorporated. 2011. *Page on denso-wave website*. Accessed on 7 January 2012. Retrieved from http://www.denso-wave.com/qrcode/index-e.html

Franeková M. 2003. *Simulácia vlastností RS kódov pre prenosové systémy súvisiace s bezpečnoťou v programovom prostredí MATLAB* [*Conferrence report*]. Accessed on 15 February 2012. Retrieved from http://dsp.vscht.cz/konference_matlab/matlab03/franekova.pdf

Gyroscope. 2008. *Gyroscope description*. Accessed on 12 March 2012. Retrieved from http://fyzmatik.pise.cz/76205-gyroskop.html

Hedrick C. 1988. *Routing Information Protocol* [*RCF 1058*] Accessed on 2 April 2012. Retrieved from http://tools.ietf.org/html/rfc1058

Image of gps system. 2006 *Image from webpage Navrchol.sk.* Accessed on 20 March 2012. Retrieved from http://www.navrchol.sk/obrazky/cestovatel/gpssystem.gif

Image of Gyroskop. 2008. *Image from webpage tahaky-referaty.sk.* Accessed on 12 March 2012. Retrieved from http://m1.aimg.sk/tahaky/g_26039_1976.jpg

Image of QR code scanning. 2009. *Image from webpage gothamguide.com.* Accessed on 11 March 2012. Retrieved from http://gothamguide.com/images/album/p1.png

Image of wifi access points. 2012. *Image from webpage AndriodPortal.sk.* Accessed on 13 March 2012. Retrieved from http://static4.androidportal.sk/wp-content/uploads/2012/02/41408898_wi_fi_inf416.gif

ISO Standardization organization 2000. *ISO/IEC 18004 - Information technology - Automatic identification and data capture techniques - Bar code symbology - QR Code*. Accessed on 21 February 2012. Retrieved from http://raidenii.net/files/datasheets/misc/qr_code.pdf

KRUPA, M. 2011. *Dekódovanie čiarového kódu v obraze v reálnom čase,* [*Master's thesis*]. BRNO. Accessed on 12 February 2012. Retrieved form https://www.vutbr.cz/studium/zaverecne-prace?zp_id=42462

Hung M., Saragas D. & Webb N. 2009. *Indoor Navigation System for Handheld Devices* [*Project report*]. Accessed on 13 February 2012. Retrieved from http://www.wpi.edu/Pubs/E-project/Available/E-project-102209-164024/unrestricted/Indoor_Navigation_System_for_Handheld_Devices.pdf

Lester, P. 2005. *A* Pathfinding for Beginners*. Accessed on 11 March 2012. Retrieved from http://www.policyalmanac.org/games/aStarTutorial.htm

Lin, J. Á., Smith, P., & Wehrle, K. 2011. *FootPath: Accurate Map-based Indoor Navigation Using Smartphones* [*Conferrence paper*]. Accessed on 10 March 2012. Retrieved from http://www.comsys.rwth-aachen.de/fileadmin/papers/2011/2011-IPIN-bitsch-footpath.pdf

Matulová, N. 2009. *Grafy a grafové algoritmy* [*Bachelor's thesis*]. Accessed on 15 March 2012. Retrieved from http://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=16188

Palúch, S. 2008. *Algoritmická teória grafov.* Accessed on 7 January 2010. Retrieved from http://frcatel.fri.uniza.sk/users/paluch/grafy.pdf

QRBCN. 2008. S*ynthesis journal 2008.* Accessed on 22 February 2012. Retrieved from http://qrbcn.com/imatgesbloc/Three_QR_Code.pdf

RFID Tags - Radio Frequency Identification Tags. 2009. [*Report of RFID technology*] Accessed on 25 March 2012. Retrieved from http://www.rfident.org/

Work, E. 2003. *UW Campus Navigator: WiFi Navigation.*[*Report of project result*] Accessed on 5 March 2012. Retrieved from http://uwcampusnav.sourceforge.net/WifiNav.pdf

ZXing library. 2007. *Open source project ZXing ("zebra crossing")*. Accessed on 12 January 2012. Retrieved from https://github.com/zxing

APPENDIX 1 ELECTRONICAL MATERIALS

Resultant applications, the map of the building Faculty of Management Science and Informatics created by application FriBasicMapEditor, JavaDoc documentation.

APPENDIX 2 QR CODES MAXIMUM DATA LIMITATION

Maximum data limitation of QR codes based on version of code (capacity are expressed by numbers of code characters)

Version	Modules	ECC Level	Data bits	Numeric	Alfanumeric	Binary	Kanji
1	21x21	L	152	41	25	17	10
		M	128	34	20	14	8
		Q	104	27	16	11	7
		H	72	17	10	7	4
2	25x25	L	272	77	47	32	20
		M	224	63	38	26	16
		Q	176	48	29	20	12
		H	128	34	20	14	8
3	29x29	L	440	127	77	53	32
		M	352	101	61	42	26
		Q	272	77	47	32	20
		H	208	58	35	24	15
4	33x33	L	640	187	114	78	48
		M	512	149	90	62	38
		Q	384	111	67	46	28
		H	288	82	50	34	21
5	37x37	L	864	255	154	106	65
		M	688	202	122	84	52
		Q	496	144	87	60	37
		H	368	106	64	44	27
6	41x41	L	1 088	322	195	134	82
		M	864	255	154	106	65
		Q	608	178	108	74	45
		H	480	139	84	58	36
7	45x45	L	1 248	370	224	154	95
		M	992	293	178	122	75
		Q	704	207	125	86	53
		H	528	154	93	64	39
8	49x49	L	1 552	461	279	192	118
		M	1 232	365	221	152	93
		Q	880	259	157	108	66
		H	688	202	122	84	52

Version	Modules	ECC Level	Data bits	Numeric	Alfanumeric	Binary	Kanji
9	53x53	L	1 856	552	335	230	141
		M	1 456	432	262	180	111
		Q	1 056	312	189	130	80
		H	800	235	143	98	60
10	57x57	L	2 192	652	395	271	167
		M	1 728	513	311	213	131
		Q	1 232	364	221	151	93
		H	976	288	174	119	74
11	61x61	L	2 592	772	468	321	198
		M	2 032	604	366	251	155
		Q	1 440	427	259	177	109
		H	1 120	331	200	137	85
12	65x65	L	2 960	883	535	367	226
		M	2 320	691	419	287	177
		Q	1 648	489	296	203	125
		H	1 264	374	227	155	96
13	69x69	L	3 424	1 022	619	425	262
		M	2 672	796	483	331	204
		Q	1 952	580	352	241	149
		H	1 440	427	259	177	109
14	73x73	L	3 688	1 101	667	458	282
		M	2 920	871	528	362	223
		Q	2 088	621	376	258	159
		H	1 576	468	283	194	120
15	77x77	L	4 184	1 250	758	520	320
		M	3 320	991	600	412	254
		Q	2 360	703	426	292	180
		H	1 784	530	321	220	136
16	81x81	L	4 712	1 408	854	586	361
		M	3 624	1 082	656	450	277
		Q	2 600	775	470	322	198
		H	2 024	602	365	250	154
17	85x85	L	5 176	1 548	938	644	397
		M	4 056	1 212	734	504	310
		Q	2 936	876	531	364	224
		H	2 264	674	408	280	173
18	89x89	L	5 768	1 725	1 046	718	442
		M	4 504	1 346	816	560	345
		Q	3 176	948	574	394	243
		H	2 504	746	452	310	191
19	93x93	L	6 360	1 903	1 153	792	488
		M	5 016	1 500	909	624	384
		Q	3 560	1 063	644	442	272
		H	2 728	813	493	338	208
20	97x97	L	6 888	2 061	1 249	858	528
		M	5 352	1 600	970	666	410
		Q	3 880	1 159	702	482	297
		H	3 080	919	557	382	235

Version	Modules	ECC Level	Data bits	Numeric	Alfanumeric	Binary	Kanji
21	101x101	L	7 456	2 232	1 352	929	572
		M	5 712	1 708	1 035	711	438
		Q	4 096	1 224	742	509	314
		H	3 248	969	587	403	248
22	105x105	L	8 048	2 409	1 460	1 003	618
		M	6 256	1 872	1 134	779	480
		Q	4 544	1 358	823	565	348
		H	3 536	1 056	640	439	270
23	109x109	L	8 752	2 620	1 588	1 091	672
		M	6 880	2 059	1 248	857	528
		Q	4 912	1 468	890	611	376
		H	3 712	1 108	672	461	284
24	113x113	L	9 392	2 812	1 704	1 171	721
		M	7 312	2 188	1 326	911	561
		Q	5 312	1 588	963	661	407
		H	4 112	1 228	744	511	315
25	117x117	L	10 208	3 057	1 853	1 273	784
		M	8 000	2 395	1 451	997	614
		Q	5 744	1 718	1 041	715	440
		H	4 304	1 286	779	535	330
26	121x121	L	10 960	3 283	1 990	1 367	842
		M	8 496	2 544	1 542	1 059	652
		Q	6 032	1 804	1 094	751	462
		H	4 768	1 425	864	593	365
27	125x125	L	11 744	3 514	2 132	1 465	902
		M	9 024	2 701	1 637	1 125	692
		Q	6 464	1 933	1 172	805	496
		H	5 024	1 501	910	625	385
28	129x129	L	12 248	3 669	2 223	1 528	940
		M	9 544	2 857	1 732	1 190	732
		Q	6 968	2 085	1 263	868	534
		H	5 288	1 581	958	658	405
29	133x133	L	13 048	3 909	2 369	1 628	1 002
		M	10 136	3 035	1 839	1 264	778
		Q	7 288	2 181	1 322	908	559
		H	5 608	1 677	1 016	698	430
30	137x137	L	13 880	4 158	2 520	1 732	1 066
		M	10 984	3 289	1 994	1 370	843
		Q	7 880	2 358	1 429	982	604
		H	5 960	1 782	1 080	742	457
31	141x141	L	14 744	4 417	2 677	1 840	1132
		M	11 640	3 486	2 113	1 452	894
		Q	8 264	2 473	1 499	1 030	634
		H	6 344	1 897	1 150	790	486
32	145x145	L	15 640	4 686	2 840	1 952	1 201
		M	12 328	3 693	2 238	1 538	947
		Q	8 920	2 670	1 618	1 112	684
		H	6 760	2 022	1 226	842	518

Version	Modules	ECC Level	Data bits	Numeric	Alfanumeric	Binary	Kanji
33	149x149	L	16 568	4 965	3 009	2 068	1 273
		M	13 048	3 909	2 369	1 628	1 002
		Q	9 368	2 805	1 700	1 168	719
		H	7 208	2 157	1 307	898	553
34	153x153	L	17 528	5 253	3 183	2 188	1 347
		M	13 800	4 134	2 506	1 722	1 060
		Q	9 848	2 949	1 787	1 228	756
		H	7 688	2 301	1 394	958	590
35	157x157	L	18 448	5 529	3 351	2 303	1 417
		M	14 496	4 343	2 632	1 809	1 113
		Q	10 288	3 081	1 867	1 283	790
		H	7 888	2 361	1 431	983	605
36	161x161	L	19 472	5 836	3 537	2 431	1 496
		M	15 312	4 588	2 780	1 911	1 176
		Q	10 832	3 244	1 966	1 351	832
		H	8 432	2 524	1 530	1 051	647
37	165x165	L	20 528	6 153	3 729	2 563	1 577
		M	15 936	4 775	2 894	1 989	1 224
		Q	11 408	3 417	2 071	1 423	876
		H	8 768	2 625	1 591	1 093	673
38	169x169	L	21 616	6 479	3 927	2 699	1 661
		M	16 816	5 039	3 054	2 099	1 292
		Q	12 016	3 599	2 181	1 499	923
		H	9 136	2 735	1 658	1 139	701
39	173x173	L	22 496	6 743	4 087	2 809	1 729
		M	17 728	5 313	3 220	2 213	1 362
		Q	12 656	3 791	2 298	1 579	972
		H	9 776	2 927	1 774	1 219	750
40	177x177	L	23 648	7 089	4 296	2 953	1 817
		M	18 672	5 596	3 391	2 331	1 435
		Q	13 328	3 993	2 420	1 663	1 024
		H	10 208	3 057	1 852	1 273	784