Indoor Navigation System Editor

Master Thesis
in

Computer Science
FernUniversität in Hagen

written by
Wei Engelbrecht
(matriculation number 8079544)

Aachen, 14th, March 2013
Statutory declaration

I herewith declare that I have completed the present thesis independently making use only of the specified literature and aids. Sentences or parts of sentences quoted literally are marked as quotations; identification of other references with regard to the statement and scope of the work is quoted. The thesis in this form or in any other form has not been submitted to an examination body and has not been published.

Date:_______________________      Signature:______________________

This work was supervised by
  • Dr. Daniela Keller
  • Dr. David Buttgereit

This work was created in cooperation with the engineering company XGraphic.
Table of Contents

1 Introduction ......................................................................................................................... 1
   1.1 Motivation ....................................................................................................................... 1
   1.2 Description of the remaining chapters ........................................................................... 1

2 Definition of terms ............................................................................................................. 2
   2.1 Indoor Navigation vs. Outdoor Navigation ................................................................. 2
   2.2 Indoor navigation system definition ............................................................................... 2
      2.2.1 Analog indoor navigation system ............................................................................ 2
      2.2.2 Digital indoor navigation system ............................................................................ 3
      2.2.3 Graph definition ...................................................................................................... 3
      2.2.4 Adjacency relation .................................................................................................. 3
      2.2.5 Distinction in terms of the graph definition ............................................................ 3
         2.2.5.1 Undirected graph ............................................................................................. 4
         2.2.5.2 Directed graph .................................................................................................. 4
         2.2.5.3 Weighted graph ............................................................................................... 4

3 “Indoor navigation system” introduction ......................................................................... 5
   3.1 User requirement analysis ............................................................................................. 5
      3.1.1 The layout files .......................................................................................................... 5
      3.1.2 The quantity and position of the software terminals ................................................ 5
      3.1.3 Change requirement ................................................................................................ 6
   3.2 A concrete application example .................................................................................... 6

4 Software development for Windows Applications with WPF and C# ............................ 8
   4.1 C# introduction .............................................................................................................. 8
   4.2 WPF introduction .......................................................................................................... 9
   4.3 .NET Framework Platform Architecture ........................................................................ 11
   4.4 AvanlonDock ................................................................................................................ 12
   4.5 Ribbon Button ............................................................................................................... 13
   4.6 3D Visualization Libraries: X3D and OpenGL .............................................................. 14

5 Algorithms ......................................................................................................................... 16
   5.1 Shortest path problem ................................................................................................... 16
   5.2 Dijkstra ......................................................................................................................... 16
      5.2.1.1 Introduction ......................................................................................................... 16
      5.2.1.2 Algorithm ............................................................................................................ 17
   5.3 Point-In-Polygon Algorithm ......................................................................................... 18
   5.4 Polygon Collision Problem ........................................................................................... 19

6 Indoor navigation system editor ....................................................................................... 21
   6.1 Basic features ................................................................................................................. 21
      6.1.1 Import 3D Models from different data sources ....................................................... 21
      6.1.2 Semi-automatic conversion 3D room-design and graph-design into 2D ............... 21
      6.1.3 Floor and rooms design .......................................................................................... 22
      6.1.4 Graph design .......................................................................................................... 22
      6.1.5 Person design .......................................................................................................... 23
6.2 Graphic User Interface (GUI) design.................................................................24
6.3 User control design.......................................................................................25
6.4 Editing and view mode..................................................................................27

7 Implementation details - Editor......................................................................29
  7.1 Class Definitions.........................................................................................29
  7.2 Main Window Functions.............................................................................31
  7.3 The precondition for main window's functions.........................................41
  7.4 Room editor...............................................................................................41
  7.5 Graph editor...............................................................................................46
  7.6 Person editor...............................................................................................51

8 Property user controls design.........................................................................53
  8.1 Floor Properties user control......................................................................53
  8.2 Graph Properties user control....................................................................57
  8.3 Person Properties user control....................................................................59
  8.4 Switch between floor- and person-property windows...............................62

9 Conclusions....................................................................................................62
  9.1 Statement of the result................................................................................62
  9.2 Open issues.................................................................................................62
    9.2.1 GUI refinement....................................................................................62
      9.2.1.1 More friendly user experience......................................................62
      9.2.1.2 Menu optimization.......................................................................63
      9.2.1.3 Multilanguage support.................................................................63
    9.2.2 Improvement of the functions in properties user controls..................63
  9.3 Outlook .......................................................................................................64
    9.3.1 Output file in 3D................................................................................64
    9.3.2 Mobile client.......................................................................................65

10 Annex – Building.xml....................................................................................66
1 Introduction

1.1 Motivation

The XGraphic engineering company develops customized, interactive Visitor- and Guidance systems for buildings, infrastructures and facilities. Such an indoor navigation system can be installed as a kiosk system in entrance halls or foyers which works as a „virtual receptionist“. Additionally it can support a digital labeling system, which uses flat touch screens as operation element at the predestined positions in the building.

An interactive visualization of 2D floor plans or 3D Model of the corresponding building is the central part of this indoor navigation system. The user can query persons, rooms or events by their names which can be inputted in a text-box. This navigation goal then is marked and the corresponding optimal path is clearly shown in the 3D model, when necessary through more than one floor.

The preparation of geometry- and building data is currently offered as a service from XGraphic engineering company and requires partly time-consuming processing according to the quality of the available data. Also it happens quite often that the building is rearranged or extended so that the floors and rooms could be partly changed as well. In a new software version this preparation should be simplified, so that the customer of this indoor navigation system can establish and maintain the geometry- and building data without specific IT knowledge.

The aim of this master thesis is the design and the implementation of a geometry and data editor for graphical editing of the models and configurations of the data contents in such an indoor navigation system.

1.2 Description of the remaining chapters

In Chapter 2 the definition of terms which are involved in these master thesis is introduced. In Chapter 3 the software „indoor navigation system“ is demonstrated with a concrete application example. In Chapter 4 the software developers platform and language which are used in this master thesis are presented. In chapter 5 the important algorithms which are used in this master thesis are explained. Chapter 6 states the basic features and working principles of the indoor navigation system editor. Chapter 7 explains the implementation details of the editor. Chapter 8 describes the functions of three Property user controls in detail. Chapter 9 summarizes the result of the editor, lists some open issues, and gives an outlook in the future for possible extensions and applications of the software.
2 Definition of terms

2.1 Indoor Navigation vs. Outdoor Navigation

A navigation system is a (usually electronic) system that aids in navigation.

*The Global Positioning System (GPS) is the most used technology for outdoor navigation. GPS is a space-based satellite navigation system that provides location and time information in all weather, anywhere on or near the earth, where there is an unobstructed line of sight to four or more GPS satellites. It is maintained by the United States government and is freely accessible to anyone with a GPS receiver.*[PS01]

But GPS has an obvious disadvantage, that is: it can not be used for indoor navigation because the GPS signals can not pass through the solid walls sufficiently. And the civil precision of GPS is not accurate enough (about 10 meter) in respect to the requirements of the precision of indoor navigation (1 meter).

In view of the above reasons there are several alternative methods used for indoor navigation. Some of them are:

- *WLAN*
- *wireless sensor-network*
- *RFID (radio-frequency identification)*
- *Bar-codes/QR codes*[DA00]

The technologies mentioned above are used mostly in mobile indoor navigation system. That means, the user walks around in the building with the help of a wireless device for positioning and navigation.

This thesis handles in contrary to mobile indoor navigation system stationary indoor navigation system. A stationary indoor navigation system is such a system in which there are several query positions. These positions are fixed in the building at certain places. When a user comes to a position, the system takes it for granted that the starting point is this position so that he only needs to enter or search the destination.

From now on when the term „indoor navigation system“ is used in this thesis, it refers to the term stationary indoor navigation system.

2.2 Indoor navigation system definition

2.2.1 Analog indoor navigation system

Analog indoor navigation system means that the navigation information is shown by signs, labels, pictogram etc. A good example of analog indoor navigation system is a shopping mall. Usually besides the escalators there are information tables that inform the customers where to find what. With the upcoming of computer and networks such an analog indoor navigation system is being replaced by digital indoor navigation system.


2.2.2 Digital indoor navigation system

The digital indoor navigation system provides the same function as an analog indoor navigation system with the support of computers and LAN/WLAN. With the help of computers and LAN/WLAN it enhances the efficiency and simplifies the exchange of information. An electronic LCD door nameplate can be easier kept up-to-date as a traditional door nameplate, similarly an electronic information table costs less to modify as a traditional information table.

2.2.3 Graph definition

The Graph concept is used to define the routes in the building. In this section the definition of Graph will be explained. The following contents are quoted from [BV01].

**Definition 2.1: Graph**

A graph is an ordered pair \( G = (V, E) \) comprising a set \( V \) of vertices or nodes\(^1\) together with a set \( E \) of edges or lines, which are 2-element subsets of \( V \) (i.e. an edge is related with two vertices, and the relation is represented as unordered pair of the vertices with respect to the particular edge). To avoid ambiguity, this type of graph may be described precisely as undirected and simple.\(^2\)

The vertices belonging to an edge are called the ends, endpoints, or end vertices of the edge. A vertex may exist in a graph and not belong to an edge.

\( V \) and \( E \) are usually taken to be finite, and many of the well-known results are not true (or are rather different) for infinite graphs because many of the arguments fail in the infinite case. The order of a graph is \( |V| \) (the number of vertices). A graph's size is \( |E| \), the number of edges.

For an edge \( \{u, v\} \), graph theorists usually use the somewhat shorter notation \( uv \).

![Diagram of a graph with 6 nodes and 7 edges.](image)

2.2.4 Adjacency relation

**Definition 2.2: Adjacency relation**

The edges \( E \) of an undirected graph \( G \) induce a symmetric binary relation \( \sim \) on \( V \) that is called the adjacency relation of \( G \). Specifically, for each edge \( \{u, v\} \) the vertices \( u \) and \( v \) are said to be adjacent to one another, which is denoted \( u \sim v \).

2.2.5 Distinction in terms of the graph definition

As stated above, in different contexts it may be useful to define the term graph with different degrees of generality. Whenever it is necessary to draw a strict distinction, the following terms are used. Most commonly, in modern texts in graph theory, unless stated otherwise, graph means "undirected simple finite graph" (see the definitions below).

---

1 In program the term nodes is used.
2 In one more generalized notion, \( E \) is a set together with a relation of incidence that associates with each edge two vertices. In another generalized notion, \( E \) is a multiset of unordered pairs of (not necessarily distinct) vertices. Many authors call this type of object a multigraph or pseudograph.
2.2.5.1 Undirected graph

Definition 2.3: Undirected graph
An undirected graph is one in which edges have no orientation. The edge \((a, b)\) is identical to the edge \((b, a)\), i.e., they are not ordered pairs, but sets \([u, v]\) (or 2-multisets) of vertices.

2.2.5.2 Directed graph

Definition 2.4: Directed graph
A directed graph or digraph is an ordered pair \(D = (V, A)\) with

- \(V\) a set whose elements are called vertices or nodes, and
- \(A\) a set of ordered pairs of vertices, called arcs, directed edges, or arrows.

An arc \(a = (x, y)\) is considered to be directed from \(x\) to \(y\); \(y\) is called the head and \(x\) is called the tail of the arc; \(y\) is said to be a direct successor of \(x\), and \(x\) is said to be a direct predecessor of \(y\). If a path leads from \(x\) to \(y\), then \(y\) is said to be a successor of \(x\) and reachable from \(x\), and \(x\) is said to be a predecessor of \(y\). The arc \((y, x)\) is called the arc \((x, y)\) inverted.

A directed graph \(D\) is called symmetric if, for every arc in \(D\), the corresponding inverted arc also belongs to \(D\).

A symmetric loopless directed graph \(D = (V, A)\) is equivalent to a simple undirected graph \(G = (V, E)\), where the pairs of inverse arcs in \(A\) correspond 1-to-1 with the edges in \(E\); thus the edges in \(G\) number \(|E| = |A|/2\), or half the number of arcs in \(D\).

A variation on this definition is the oriented graph, in which at most one of \((x, y)\) and \((y, x)\) may be arcs.

2.2.5.3 Weighted graph

Definition 2.5: Weighted graph
A graph is a weighted graph if a number (weight) is assigned to each edge. Such weights might represent, for example, costs, lengths or capacities, etc. depending on the problem at hand. Some authors call such a graph a network.

➢ The „indoor navigation system“ developed by XGraphic engineering company and its editor developed for the sake of this master thesis use a weighted symmetric directed graph for the routes definition in the building. The weight of every edge (vector) is the distance between the start node and end node.
3 “Indoor navigation system” introduction

The XGraphic engineering company has developed applications of „indoor navigation system“ for several different customers. In this chapter the basic ideas of this software will be introduced.

3.1 User requirement analysis

3.1.1 The layout files

First of all the layout of the building must be defined. Usually the customers have:

- CAD 3D data model for the whole building

  AutoCAD is the most popular software application for computer-aided design (CAD) and drafting. The software supports both 2D and 3D formats. The software is developed and sold by Autodesk, Inc.

  ➢ As the output file from AutoCAD the building layout definition has a .DWG file-format. With the help of additional converters a .DWG file can be converted into DXF (Drawing Interchange Format, or Drawing Exchange Format) format which enables data interoperability between AutoCAD and other programs.

- 2D picture file for each floor

  Some customers have instead of CAD-DXF file a set of 2D picture files. These picture files can be in one of *.bmp, *.gif, *.jpeg, *.png, *.tiff, *.jpg, *.ico file-format, and each file contains one floor's 2D layout.

  Whether a 3D CAD-DXF file or a set of 2D picture files, the „indoor navigation system“ has to convert them to a special BIN file as a standard input data model.

3.1.2 The quantity and position of the software terminals

The customer must specify the quantity and position of the software terminals. Each terminal has a standpoint which is defined as a query node of the whole building graph. A terminal can be a computer which is located usually at the important points of the building with keyboard or touch screen. Besides that digital LCD door nameplates are used at each room to inform which persons work or which activity is going to be in that room. As a precondition all terminals and digital LCD door nameplates must be connected to the indoor navigation server via LAN or WLAN.

---

3 The detailed explanation is written in 4.6 3D Visualization Libraries: X3D and OpenGL
3.1.3 Change requirement

Once the floors' layout and the whole building's graph are defined, they are not immutable and frozen. The floor-layouts need to be changed frequently. For example, a wall between 2 rooms is removed; or a room is divided into 2 parts. It can also happen that the building is extended with a new floor or shrinks to a smaller floor. Last but not least persons or equipments can be relocated in the building. Due to this continuous change a simple way to edit the floor's layout and the related graph is unavoidable. In the first version of the “indoor navigation system” all the floor layouts and graphs are implemented by the XGraphic engineering company. This takes too much time and of course also brings additional costs to the customers. That is the reason to develop an indoor navigation system editor.

3.2 A concrete application example

- RWTH main building

The main building of RWTH Aachen University is a very good example. It has installed the indoor navigation system from XGraphic engineering company since February 2006. The system can display the floor-layouts in 3D format or in 2D format on demand of the user. The user can search rooms, persons, auditoriums and institutes as destination by inputting their names, accordingly the system will show the optimal detailed graphic route-description to guide the user. This system is intensively used by the students and the visitors and has got a good reputation. Now the RWTH Aachen University is considering the system extension. The first intention is to extend the system from one building based to inter building based, the system should deliver a complete solution for the campus and a combination of indoor and outdoor navigation system as well.
3. “Indoor navigation system” introduction

**Drawing 6:** RWTH main building - 3D floor layout

**Drawing 7:** RWTH main building: 2D route description

**Drawing 8:** RWTH main building: simple 2D line-drawing
4 Software development for Windows Applications with WPF and C#

The XGraphic engineering company has chosen WPF UI Framework and C# language for the development of the indoor navigation system editor. The development environment is Microsoft Visual Studio 2010. Additionally the WPF control library AvanlonDock, the UI interface Ribbon Button and the self-made 3D visualization library X3D which is based on OpenGL are implemented in the software development. In this chapter all these development tools are introduced.

4.1 C# introduction

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. We can use C# to create traditional Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much more. Visual C# provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on version 4.0 of the C# language and version 4.0 of the .NET Framework.

C# syntax is highly expressive, yet it is also simple and easy to learn. The curly-brace syntax of C# will be instantly recognizable to anyone familiar with C, C++ or Java. Developers who know any of these languages are typically able to begin to work productively in C# within a very short time. C# syntax simplifies many of the complexities of C++ and provides powerful features such as nullable value types, enumerations, delegates, lambda expressions and direct memory access. C# supports generic methods and types, which provide increased type safety and performance, and iterators, which enable implementers of collection classes to define custom iteration behaviors that are simple to use by client code. Language-Integrated Query (LINQ) expressions make the strongly-typed query a first-class language construct.

As an object-oriented language, C# supports the concepts of encapsulation, inheritance, and polymorphism. All variables and methods, including the Main method, the application's entry point, are encapsulated within class definitions. A class may inherit directly from one parent class, but it may implement any number of interfaces. Methods that override virtual methods in a parent class require the override keyword as a way to avoid accidental redefinition. In C#, a struct is like a lightweight class; it is a stack-allocated type that can implement interfaces but does not support inheritance.

In addition to these basic object-oriented principles, C# makes it easy to develop software components through several innovative language constructs, including the following:

- Encapsulated method signatures called delegates, which enable type-safe event notifications.
- Properties, which serve as accessors for private member variables.
- Attributes, which provide declarative metadata about types at run time.
- Inline XML documentation comments.
- Language-Integrated Query (LINQ) which provides built-in query capabilities across a variety of data sources.[MS04]
4. Software development for Windows Applications with WPF and C#

4.2 WPF introduction

Developed by Microsoft, the Windows Presentation Foundation (or WPF) is a computer-software graphical subsystem for rendering user interfaces in Windows-based applications. WPF, previously known as "Avalon", was initially released as part of .NET Framework 3.0. Rather than relying on the older GDI subsystem, WPF utilizes DirectX. WPF attempts to provide a consistent programming model for building applications and provides a separation between the user interface and the business logic. It resembles similar XML-oriented object models, such as those implemented in XUL and SVG.

WPF employs XAML, an XML-based language, to define and link various user interface (UI) elements. WPF applications can also be deployed as standalone desktop programs, or hosted as an embedded object in a website. WPF aims to unify a number of common user interface elements, such as 2D/3D rendering, fixed and adaptive documents, typography, vector graphics, runtime animation, and pre-rendered media. These elements can then be linked and manipulated based on various events, user interactions, and data bindings.

WPF runtime libraries are included with all versions of Microsoft Windows since Windows Vista and Windows Server 2008.

As of 2011 Microsoft has released four major WPF versions: WPF 3.0 (Nov 2006), WPF 3.5 (Nov 2007), WPF 3.5sp1 (Aug 2008), and WPF 4 (April 2010).

➢ The development of the indoor navigation system editor uses WPF4.

The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware. WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text, and typography. WPF is included in the Microsoft .NET Framework, so the WPF applications can be built which incorporate other elements of the .NET Framework class library.

WPF exists as a subset of .NET Framework types that are for the most part located in the System.Windows namespace. The fundamental WPF programming experience is quite similar as the .NET Framework (ASP.NET and Windows Forms). With WPF we can instantiate classes, set properties, call methods and handle events.

To support some of the more powerful WPF capabilities and to simplify the programming experience, WPF includes additional programming constructs that enhance properties and events: dependency properties, data binding and routed events.

- Dependency properties

  The purpose of dependency properties is to provide a way to compute the value of a property based on the value of other inputs. These other inputs might include system properties such as themes and user preference, just-in-time property determination mechanisms such as data binding and animations/storyboards, multiple-use templates such as resources and styles, or values known through parent-child relationships with other elements in the element tree. In addition, a dependency property can be implemented to provide self-contained validation, default values, callbacks that monitor changes to other properties, and a system that can coerce property values based on potentially runtime information.
• Data Binding

Most applications are created to provide users with the means to view and edit data. For WPF applications, the work of storing and accessing data is already provided for by technologies such as Microsoft SQL Server and ADO.NET. After the data is accessed and loaded into an application's managed objects, the hard work for WPF applications begins. Essentially, this involves two things:

1. Copying the data from the managed objects into controls, where the data can be displayed and edited.
2. Ensuring that changes made to data by using controls are copied back to the managed objects.

To simplify application development, WPF provides a data binding engine to automatically perform these steps. The core unit of the data binding engine is the Binding class, whose job is to bind a control (the binding target) to a data object (the binding source). This relationship is illustrated by the following figure.[MS06]

**Drawing 9: Data Binding[MS06]**

• Routed events

A routed event is a common language runtime (CLR) event that is backed by an instance of the RoutedEventArgs class and registered with the WPF event system. The RoutedEventArgs instance obtained from registration is typically retained as a public static readonly field member of the class that registers and thus "owns" the routed event. The connection to the identically named CLR event (which is sometimes termed the "wrapper" event) is accomplished by overriding the add and remove implementations for the CLR event. Ordinarily, the add and remove are left as an implicit default that uses the appropriate language-specific event syntax for adding and removing handlers of that event. The routed event backing and connection mechanism is conceptually similar to how a dependency property is a CLR property that is backed by the DependencyProperty class and registered with the WPF property system.

Routed events use one of three routing strategies:

1. **Bubbling:** Event handlers on the event source are invoked. The routed event then routes to successive parent elements until reaching the element tree root. Most routed events use the bubbling routing strategy. Bubbling routed events are generally used to report input or state changes from distinct controls or other UI elements.

2. **Direct:** Only the source element itself is given the opportunity to invoke handlers in response. This is analogous to the "routing" that Windows Forms uses for events. However,

---

4 An application's managed object is an object which is manipulated by the program.
unlike a standard CLR event, direct routed events support class handling (class handling is explained in an upcoming section) and can be used by EventSetter and EventTrigger.

3. **Tunneling**: Initially, event handlers at the element tree root are invoked. The routed event then travels a route through successive child elements along the route, towards the node element that is the routed event source (the element that raised the routed event). Tunneling routed events are often used or handled as part of the compositing for a control, such that events from composite parts can be deliberately suppressed or replaced by events that are specific to the complete control.

An application developer does not always need to know or care that the event they are handling is implemented as a routed event. Routed events have special behavior, but that behavior is largely invisible if the developers are handling an event on the element where it is raised. [MS03]

○ An example for adding an event handler in XAML
  `<Button Click="b1SetColor">button</Button>`

○ An example for adding an event handler in C# codes
  ```csharp
  void MakeButton2()
  {
    Button b2 = new Button();
    b2.Click += new RoutedEventHandler(Onb2Click2);
  }
  
  void Onb2Click2(object sender, RoutedEventArgs e)
  {
    //logic to handle the Click event
  }
  ```

4.3 **.NET Framework Platform Architecture**

C# programs run on the .NET Framework, an integral component of Windows that includes a virtual execution system called the common language runtime (CLR) and a unified set of class libraries. The CLR is the commercial implementation by Microsoft of the common language infrastructure (CLI), an international standard that is the basis for creating execution and development environments in which languages and libraries work together seamlessly.

Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code and resources, such as bitmaps and strings, are stored on disk in an executable file called an assembly, typically with an extension of .exe or .dll. An assembly contains a manifest that provides information about the assembly's types, version, culture, and security requirements.
When the C# program is executed, the assembly is loaded into the CLR, which might take various actions based on the information in the manifest. Then, if the security requirements are met, the CLR performs just in time (JIT) compilation to convert the IL code to native machine instructions. The CLR also provides other services related to automatic garbage collection, exception handling, and resource management. Code that is executed by the CLR is sometimes referred to as "managed code," in contrast to "unmanaged code" which is compiled into native machine language that targets a specific system. The following diagram illustrates the compile-time and run-time relationships of C# source code files, the .NET Framework class libraries, assemblies, and the CLR.[MS04]

**4.4 AvalonDock**

AvalonDock is a WPF controls library which can be used to create a docking layout system like that is present in VisualStudio. It supports fly-out panes, floating windows, multiple docking manager in same window, styles and themes and it can host WinForms controls.

**Features**

- Supports Model-View-ViewModel (MVVM) design
- Almost everything can be restyled
- Support for Windows Forms controls[AD00]
In GUI-based application software, a ribbon is an interface where a set of toolbars are placed on tabs in a tab bar. Recent releases of some Microsoft and Autodesk applications have popularized this form with a modular ribbon as their main interface. A ribbon provides the user interface of an application with a large toolbar filled with graphical representations of control elements which are grouped by different functionality. Ribbons can contain tabs to expose different sets of control elements, eliminating the need for many different icon-based tool bars. Some of these tabs are contextual and appear only when a certain type of object is selected, providing specific tools for items such as tables or images.[WK09]

The Ribbon is a way to organize related commands so that they are easier to find. Commands appear as controls on the Ribbon. Controls are organized into groups along a horizontal strip at the top edge of an application window. Related groups are organized on tabs. Most of the features that were accessed by using menus and toolbars in earlier versions of the Microsoft system can now be accessed by using the Ribbon.[MS05]
4.6 3D Visualization Libraries: X3D and OpenGL

In order to show 3D graphical models in Windows application, in principle there are only 2 libraries that can be used: DirectX and OpenGL.

- **Microsoft DirectX** is a collection of application programming interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms. Originally, the names of these APIs all began with Direct, such as Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectSound, and so forth. [MH01]

- **OpenGL (Open Graphics Library)** is a cross-language, multi-platform API for rendering 2D and 3D computer graphics. The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering. OpenGL was developed by Silicon Graphics Inc. in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games. OpenGL is managed by the non-profit technology consortium Khronos Group. [RW01]

The indoor navigation system uses OpenGL for the 3D visualization.

Drawing 13 illustrates the 3D Visualization Model of the indoor navigation system.

X3D is a OCX file written in C++ which uses OpenGL Library for 3D
Software development for Windows Applications with WPF and C#

4. Software development for Windows Applications with WPF and C#

3D stores all the data of the Geometry which are added to it in an internal list. When X3D needs to display the 3D-Graphics it uses all kinds of different functions from the OpenGL Library to draw them on the screen. Since OpenGL can only draw points, lines, and triangles, X3D must build up the desired 3D model from a small set of geometric primitives - points, lines, and triangular polygons.

XG3D is a dynamic-link library (DLL) file which is written in C# and encapsulated X3D.OCX so that the X3D.ocx can be used in Windows Forms Applications. Again XG3DWPF is a DLL file written in C# which encapsulated XG3D.dll so that X3D.ocx can be used in Windows Presentation Foundation (WPF) Applications. For the sake of simple implementation, the XGraphic engineering company has developed interfaces between:

1. X3D.OCX and XG3D.dll
2. XG3D.dll and XG3DWPF.dll

Because this is an on-going work, not all functions in X3D.OCX have interfaces yet. E.g. the function WorldToScreen() in X3D.OCX has no interface in XG3DWPF.dll, but it is used in indoor navigation system and its editor for implementation of 3D models. 3D data sources such as AutoCAD-DXF file, Virtual Reality Modeling Language-WRL file, or Wavefront-OBJ file can be converted by a data converter to a kind of binary file, which can be used by function LoadBin() for the 3D model visualization in the system.

There is a Geometry.IO sub model in XG3D.dll which can trigger related events as soon as 3D geometry data are added. The function Loadbin(binfile) in Geometry.IO works for loading a .bin file and generating events. These events are captured by XG3D.dll and then the XG3D calls the corresponding methods in X3D to add them for displaying. We can add event handler to the XG3D which can capture the special geometry-changes and override the original functions to achieve a specific implementation.
5 Algorithms

In this chapter the algorithms which are used in the indoor navigation system editor will be explained.

5.1 Shortest path problem

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.\cite{SR00}

The shortest path problem can be defined for graphs whether undirected, directed, or mixed. It is defined here for undirected graphs; for directed graphs the definition of path requires that consecutive vertices are connected by an appropriate directed edge.

Two vertices are adjacent\(^5\) when they are both incident to a common edge. A path in an undirected graph is a sequence of vertices \(P = (v_1, v_2, ..., v_n) \in V \times V \times ... \times V\) such that \(v_i\) is adjacent to \(v_{i+1}\) for \(1 \leq i < n\). Such a path \(P\) is called a path of length \(n\) from \(v_1\) to \(v_n\). (The \(v_i\) are variables; their numbering here relates to their position in the sequence and needs not to relate to any canonical labeling of the vertices.)

Let \(e_{i,i+1}\) be the edge incident to both \(v_i\) and \(v_{i+1}\). Given a real-valued weight function \(f : E \rightarrow \mathbb{R}\), and an undirected (simple) graph \(G\), the shortest path from \(v\) to \(v'\) is the path \(P = (v_1, v_2, ..., v_n)\) (where \(v_1 = v\) and \(v_n = v'\)) that over all possible \(n\) minimizes the sum \(\sum_{i=1}^{n-1} f(e_{i,i+1})\). When the graph is unweighted or \(f : E \rightarrow \{c\}, \ c \in \mathbb{R}^+\), this is equivalent to finding the path with fewest edges.

The problem is also sometimes called the single-pair shortest path problem, to distinguish it from the following variations:

- The single-source shortest path problem, in which we have to find shortest paths from a source vertex \(v\) to all other vertices in the graph.
- The single-destination shortest path problem, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex \(v\). This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.
- The all-pairs shortest path problem, in which we have to find shortest paths between every pair of vertices \(v, v'\) in the graph. \cite{CT01}

In this thesis a solution for the single-source shortest path problem is needed.

5.2 Dijkstra

5.2.1.1 Introduction

Dijkstra’s algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is

\(^5\) For detailed explanation, see 2.2.4
often used in routing and as a subroutine in other graph algorithms. In this thesis Dijkstra’s algorithm is used to check the graph validity⁶.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra’s algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols.

Dijkstra’s original algorithm runs in $O(|V|^2)$. The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O((E + |V| \log |V|)$ is due to (Fredman & Tarjan 1984). This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.[CT01]

5.2.1.2 Algorithm

Let the node at which we are starting be called the initial node. Let the distance of node $Y$ be the distance from the initial node to $Y$. Dijkstra’s algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes unvisited. Set the initial node as current. Create a set of the unvisited nodes called the unvisited set consisting of all the nodes except the initial node.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. For example, if the current node $A$ is marked with a tentative distance of 6, and the edge connecting it with a neighbor $B$ has length 2, then the distance to $B$ (through $A$) will be $6+2=8$. If this distance is less than the previously recorded tentative distance of $B$, then overwrite that distance. Even though a neighbor has been examined, it is not marked as "visited" at this time, and it remains in the unvisited set.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again; its distance recorded now is final and minimal.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal), then stop. The algorithm has finished.
6. Set the unvisited node marked with the smallest tentative distance as the next "current node" and go back to step 3. [CT01]

⁶ For detailed explanation, see 7.2 „Check Building Graph“ section.
5.3 **Point-In-Polygon Algorithm**

By room-design, graph-design and persons-design\(^7\) the system needs to check in which room a point is located (e.g. per mouse click). The Point-In-Polygon Algorithm will be used. Drawing 15 demonstrates a typical case of a severely concave polygon with 14 sides. The red dot is a point which needs to be tested, to determine if it lies inside the polygon.

The solution is to compare each side of the polygon to the \(Y\) (vertical) coordinate of the test point, and compile a list of nodes, where each node is a point where one side crosses the \(Y\) threshold of the test point. In this example, eight sides of the polygon cross the \(Y\) threshold, while the other six sides do not. Then, if there are an odd number of nodes on each side of the test point, then it is inside the polygon; if there are an even number of

---

\(^7\) For detail information, see 6.1 Basic features.
nodes on each side of the test point, then it is outside the polygon. In this example, there are five nodes to the left of the test point, and three nodes to the right. Since five and three are odd numbers, our test point is inside the polygon.

**Algorithm**

Boolean pointInPolygon(Point p, Polygon polygon)

1. polySides ← the number of polygon's vertices
2. polyX[] ← horizontal coordinates of vertices
3. polyY[] ← vertical coordinates of vertices
4. x, y ← the coordinate of p
5. j ← polySides -1
6. boolean oddNodes ← false
7. for i ← 0 to polySides
9. if polyX[i]+(y-polyY[i])/(polyY[j]-polyY[i]) * (polyX[j]-polyX[i])<x
10. oddNodes=!oddNodes
11. j ← i
12. return oddNodes

Note: This algorithm does not care whether the polygon is traced in clockwise or counterclockwise fashion.[DR00]

5.4 Polygon Collision Problem

If the boundary of a polygon crosses the other polygon's boundary, this is called polygon collision. There are two algorithms for detection of polygon collision: algorithm for convex polygons and algorithm for nonconvex polygons. Since a room can be a nonconvex polygon, in the indoor navigation system editor only the algorithm for nonconvex polygons will be used.

The following contents are quoted form [GPWiKi00].

**Simple Nonconvex Polygons**

For nonconvex but still simple polygons, an acceptable method is to iterate over all vertices and perform the Point-in-polygon test. The advantage of this method is that we can compute the exact intersection point and collision. When there is a point that lies inside the other polygon, we just iterate over all edges of the second polygon again and look for edge intersections. Note that this method detects collision when it already happens. This algorithm is fast enough to perform it hundreds of times per sec.

**Edge intersection test (assuming Vector2D is a structure carrying the coordinates):**

1. double determinant(Vector2D vec1, Vector2D vec2){
2. return vec1.x * vec2.y - vec1.y * vec2.x;
3. }
4.
5. //one edge is a-b, the other is c-d
6. Vector2D edgeIntersection(Vector2D a, Vector2D b, Vector2D c, Vector2D d) {
    double det = determinant(b - a, c - d);
    double t   = determinant(c - a, c - d) / det;
    double u   = determinant(b - a, c - a) / det;
    if ((t < 0) || (u < 0) || (t > 1) || (u > 1)) {
        return NO_INTERSECTION;
    } else {
        return a * (1 - t) + t * b;
    }
}
6 Indoor navigation system editor

In this chapter the basic features and the graphical user interface (GUI) design of the indoor navigation system editor are introduced.

6.1 Basic features

6.1.1 Import 3D Models from different data sources

The editing platform of the indoor navigation system editor is based on 2D. As mentioned in 3.1.1, the user may have 2D or 3D building layout. For 2D layout picture, the editor can use it as the background picture for each floor's room-design, the implementation is very easy. For 3D layout picture, the editor must convert the 3D data model to several 2D pictures, which can be used as floor layouts. If in the 3D layout the routes are already designed, the editor should enable the user to import the 3D route design into the corresponding floor.

There are two possibilities to design the routes for a 3D building layout:

- 3D routes design in a 3D data model
- 2D routes design for each floor

In this thesis only the 2nd case is considered. The reasons are:

1. Customer requirement. Some customers want to have 2D indoor navigation system although they have 3D building layout.

2. With the output of 2D indoor navigation system editor it is possible to convert the 2D routes design into 3D with the help of XG3DWPF.dll. The program can be expanded as needed.

As mentioned in 4.6, 3D data sources such as AutoCAD-DXF file, Virtual Reality Modeling Language-WRL file, or Wavefront-OBJ file can be converted by a data converter to a kind of binary file, which can be used by XG3DWPF.dll for the 3D model visualization.

6.1.2 Semi-automatic conversion 3D room-design and graph-design into 2D

The indoor navigation system editor takes advantage of layer property of each 3D geometry as a filter. Layer property is widely used in 3D Model Design.

Basically, a layer is a set of objects that are classified together. This allows user to create a layer called e.g. walls and have this layer’s properties unique to the layer walls. One way to think of layers is as if you are drawing on a transparent sheet and for each item you are using a different sheet. Now with each object on a separate layer we can turn certain layers off and it has the same effect as removing that layers transparent sheet. From this example you can see how they become invaluable in large drawings. [AC00]

In the .bin file, each geometry’s layer name is also imported. The system can display the geometries from certain layers which are specified by the user. In this way the 3D room-design for a certain floor can be imported and converted into 2D room-design as well as the 3D graph-design for a

---

8 For detailed explanation, see 4.6.
certain floor.

### 6.1.3 Floor and rooms design

With the indoor navigation system editor the user can make the room-design for each floor. He can also change or delete the existing floors. By changing floor layout file the user can determine whether the old room-design should be kept or not. By deleting a floor layout file all room-design and graph-design of this floor will be deleted as well.

With the floor layout file as a background picture, the user can design rooms on it. A room is a simple nonconvex or convex polygon. The user can change any existing vertex or polygon to another position. He can also do the following actions:

- add /delete vertices to/from an existing polygon.
- Delete an existing polygon
- Edit the room's name

### 6.1.4 Graph design

In order to calculate the shortest-path between two positions in a building, the indoor navigation system editor implements graph conception for the route-design.

Graph design includes two elements-design:

- Node design.
  - a). Add node: user can add a new node by mouse click.
  - b). Edit node. This again includes:
    - i). Edit node's name
    - ii). Edit node's position: user can drag and move the node. If the node doesn't locate in any room, then the movement is invalid and the node will be automatically set back to its original position.
    - iii). Edit node's type

There are three types of nodes in the indoor navigation system editor:

**Normal node:** the nodes which can only be used to create vectors with the other nodes of the same floor.

**Floor entry node:** Besides the function of a normal node, floor entry node can be used to create inter-floor-vectors with other entry node. For each floor there must be at least one floor entry node.

**Building entry node:** Besides the function of a normal node, building entry node can be used to create inter-floor-vectors with other entry node. For each building there must be at least one building entrance node.

The difference between floor entry node and building entry node is reflected in the rules of

---

9 For detailed explanation, see 2.2.3.
10 An entry node can be a floor entry node or a building entry node.
11 The two nodes of an inter-floor-vector belong to two different floors.
6. Indoor navigation system editor

graph-validity which is explained in 7.2 „Check Building Graph“ section.
c). Delete node. By deleting a node all of its related vectors will also be deleted.

- Vector design. Vector represents the path between two nodes.
  a). Add Vector
  b). Edit Vector. A vector can be indirectly edited by editing its related nodes.
  c). Delete Vector

A vector includes three elements: start node, end node and distance.

For the distance calculation of a normal vector which two nodes are at the same floor, it uses the following formula:

Suppose it has Node1 with coordinates (x1,y1) and Node2 with coordinates (x2,y2), the distance

\[ D = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} \] .

For the distance calculation of an inter-floor vector which 2 nodes are not at the same floor\(^{12}\), it uses the following formula:

Suppose it has Node1 with coordinates (x1,y1) and Node2 with coordinates (x2,y2), and z1 is the height of the Node1's floor, z2 is the height of the Node2's floor, the distance

\[ D = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2 + (z_1-z_2)^2} \] .

As mentioned in 2.2.5.3, the indoor navigation system editor uses weighted symmetric directed graph. From Definition 2.4\(^{13}\) we can derive:

if Vector (start node NodeA, end node NodeB, distance AB) exists, then Vector (start node NodeB, end node NodeA, distance AB) must exist.

To simplify the program and to reduce unnecessary storage, only the oriented graph will be saved. By graph validity check (Dijkstra check) the weighted symmetric directed graph will be created in the program.

Instead of saving graph for each floor, the graph in the indoor navigation system editor is stored for the whole building.

6.1.5 Person design

A person can only work in one place at the same time. If the working place of a person is changed, it must be changed in the indoor navigation system editor as well.

Person design includes:

- Add person. This includes two parts:
  a). Add person's name
  b). Add person's photo: without assigning a photo to a person the system will use a default person icon to represent this person in the software.

- Edit person. This includes three parts:
a). Edit person's name: this can be done from person's property window\textsuperscript{14}.

b). Edit person's picture

c). Edit person's position: this can be done by drag and drop person's image to the right position or from person's property window directly allocate the person into one room. If the person is not located in any rooms, by saving the system will treat this allocation as invalid and the person's position keeps unassigned.

- Delete person. By deleting the person will be deleted from the whole building. But if the user only deletes the person's image from the room-design, it means only that the person's position is not assigned.

A person is only a concrete example for all activities and equipments in the building. If the customer can edit persons in the indoor navigation system, it can be easily extended to edit equipments (e.g. servers, printers, infrastructures, firefighting points, Quick Response (QR) codes etc.) and activities (e.g. meetings, seminars, etc.) as well.

6.2 Graphic User Interface (GUI) design

Drawing 16 is a screen-shot of the GUI of the indoor navigation system editor. The main window uses Ribbon Button\textsuperscript{15}, all the three subwindows have implemented AvanlonDock\textsuperscript{16} control library. At the left side of the main window is the treeview subwindow, in which all buildings are displayed. Each building is independent from other buildings. E.g. for an university it is quite normal that the user has more than one building to maintain. So this design is more convenient for the user even though there is no inter-building navigation yet. In the middle of the main window is the mainview subwindow, in which the current floor/graph user control is displayed. At the right side of the main window is the property subwindow, in which the related properties of the selected item or user control are displayed.

In the treeview sub-window each building item has five different kinds of sub items. They are:

- Floor items
- “Rooms” Folder items
- Room items
- Graph items
- Person items

\textsuperscript{14} For detailed explanation, see 8.3.
\textsuperscript{15} For detailed explanation, see 4.5.
\textsuperscript{16} For detailed explanation, see 4.4.
6.3 User control design

In the indoor navigation system editor the modular programming\textsuperscript{17} embodies in the User Control design. User Control is a subclass from System.Windows.Controls, which can be designed as a customized reusable component in Windows Presentation Foundation (WPF) application. User controls can contain other controls, resources, and animation timelines, just like a WPF application. The only difference is that the root element is a User Control instead of a Window or a Page in XAML file.\cite{MS00}

\textsuperscript{17} For detailed explanation, see 5.5.
In the indoor navigation system editor beside the main window there are six fundamental User Controls (see Table 1). Each User Control has its special functionality which can be used by the main window. The main window acts as a cooperator among the user controls. It creates User Control on demand so that the User Control can handle the data and send back the result which can be used for initialization of other User Controls by the main window. All these User Controls are programmed modularly so that they can be reused by other projects without change or with very few changes.

<table>
<thead>
<tr>
<th>User Control Name</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floor User Control</td>
<td>1. Define and modify floor layout</td>
</tr>
<tr>
<td></td>
<td>2. Define and modify rooms</td>
</tr>
<tr>
<td></td>
<td>3. Define and modify persons' positions</td>
</tr>
<tr>
<td>Graph User Control</td>
<td>1. Define and modify graph in floor</td>
</tr>
<tr>
<td></td>
<td>2. Define and modify graph in building</td>
</tr>
<tr>
<td>3D Model User Control</td>
<td>1. Display the 3D data model</td>
</tr>
<tr>
<td></td>
<td>2. Display the 3D data model according to the user's selection</td>
</tr>
<tr>
<td></td>
<td>3. Convert the top view of the 3D data model into 2D layout</td>
</tr>
<tr>
<td></td>
<td>4. Import selected lines as graph-design into 2D layout</td>
</tr>
<tr>
<td>Floor property User</td>
<td>1. Display the detailed information of the floor.</td>
</tr>
<tr>
<td>Control</td>
<td>2. Display the detailed information of the selected room.</td>
</tr>
<tr>
<td></td>
<td>3. Display the detailed information of the selected point.</td>
</tr>
<tr>
<td>Graph property User</td>
<td>1. Display the detailed information of the selected node.</td>
</tr>
<tr>
<td>Control</td>
<td>2. Set node name, node type (floor entry node or building entry node).</td>
</tr>
<tr>
<td></td>
<td>3. Link the selected entry node to other entry node of other floor.</td>
</tr>
<tr>
<td></td>
<td>4. Delete the inter-floor-vectors of the selected entry node.</td>
</tr>
<tr>
<td>Person property User</td>
<td>1. Display the detailed information of the selected person.</td>
</tr>
<tr>
<td>Control</td>
<td>2. Change selected person's name.</td>
</tr>
<tr>
<td></td>
<td>3. Allocate selected person to a certain room of a certain floor.</td>
</tr>
</tbody>
</table>

Table 1: Modular User Controls

By clicking on the treeview item in the mainview the related user control will be displayed. If for each clicking on a treeview item the system creates a new user control in the mainview, there are 2 disadvantages:

1. Waste too much time and resource for initialization of the same user controls.
2. The intermediate editing status in the user control may be lost.

To make the program more effectively, the system uses 3 user control dictionaries to save the created user controls. They are:

a. floorDictionary: to save all created floor user controls.

b. roomsDictionary: to save all created room-design user controls.

c. graphDictionary: to save all created graph user controls.

When the selected item of the treeview is changed, the system first checks the item's type, and looks into the related dictionary accordingly. If this item's ID exists in the dictionary's key list, then the value (This is a user control) of this key will be shown in mainview; if not, that means this user
control hasn’t been shown before, then a new user control instance will be created and shown in the mainview, at the same time a new KeyValuePair for the dictionary is created for possible re-display later. In the property subwindow the corresponding property user control will be displayed.

6.4 Editing and view mode

By default the user control in the mainview is in view mode, in which the user can not edit the elements in it. With appropriate preconditions explained below this user control can be switched into editing mode. To set the user control into editing mode, just select the item in the treeview sub-window, then right click. If the preconditions are fulfilled, the checkable context menu „In Editing Mode“ is enabled; otherwise this context menu is disabled (Drawing 17).

In editing mode the user can edit room-design or graph-design directly in the mainview.

Again for the sake of program efficiency, the system uses two dictionaries to save the user controls in editing mode. They are:

a. inEditingGraphDictionary: to save the graph user controls in editing mode
b. inEditingRoomsDictionary: to save the room-design user controls in editing mode.

The room-design of each floor is independent of other floors. That means, more than one floor simultaneously in editing mode is permitted. But if one floor is in editing mode, its related graph can not be switched into editing mode at the same time because the modified room-design can lead to an invalid node design. For better understanding of this point, we look at the following example:

A building has two floors, Floor1 and Floor2. In Floor1 there is a floor entry node Node1, which constitutes an inter-floor-vector with Node2 in Floor2. After the Floor1 is set into editing mode, the room in which Node1 is located maybe deleted and Node1 will be deleted as well when the user saves the changes because a node which is not located in any room is invalid. If it is permitted by the system to set the graph-design of this building into editing mode at the same time, let's say the graph of Floor2, then the system will load all inter-floor-vectors into the graph user control for later storage. Node1 will be loaded as part of inter-floor-vector because at this time point the Floor1 is in editing mode.

Drawing 17: Switch the room-design user control to editing mode

Switch to editing mode

---

18 It is determined by the data structure. For detail information, see 7.1 „class definitions“.
editing mode and the user hasn't saved the change yet. After that point of time the user sets Floor1 back to view mode and saves the changes of Floor1 room-design, Node1 is deleted from this floor but it is already loaded into Floor2 graph-design. This will cause error when the graph-design of Floor2 is set back to view mode and the user wants to save the modified graph-design.

On the other hand when the graph of one floor is in editing mode, the room-design of this floor can not be in editing mode at the same time because of the same reason above. As mentioned before that the graph is saved for the whole building, so if in one building the graph of one floor is in editing mode, the other floors' graph can not be switched into editing mode simultaneously because it may cause the error of inter-floor-vectors.

The preconditions mentioned above are the principles for the development of the indoor navigation system editor. In 7.3 „The preconditions for main window's functions“ they will be explained in more detail. By leaving the editing mode the system will save the changes after the user's confirmation (Drawing 18).

\[\text{Drawing 18: By leaving editing mode the system asks for saving changes}\]
7 Implementation details - Editor

7.1 Class Definitions

The Unified Modeling Language (UML) diagram of the class definitions is shown in Drawing 19.

- Building Class. It includes:
  1. the building ID (automatically generated by the system)
  2. the building name (cannot be empty, given by the user)
  3. a list of floor instances
  4. a graph instance
  5. a list of person instances
● Floor Class. It includes:
  1. the floor ID (automatically generated by the system)
  2. the floor name (cannot be empty, given by the user)
  3. the location of the floor's layout file
  4. a list of room instances

● Room Class. It includes:
  1. the room ID (automatically generated by the system)
  2. the room name (cannot be empty, given by the user)
  3. a list of point instances

● Point Class. It includes:
  1. the point ID (automatically generated by the system)
  2. the point coordinates x, y

● Graph Class. It includes:
  1. a list of node instances
  2. a list of vector instances

● Node Class. It includes:
  1. the node ID (automatically generated by the system)
  2. the node name (cannot be empty, given by the user)
  3. the floor ID (in which floor the node is located.)
  4. the room ID (in which room the node is located.)

If a user changes a floor's room-design after its graph-design is defined, by saving the modified room-design the system will automatically update the graph-design in which the invalid nodes will be deleted. „Invalid“ nodes are the nodes which are not located in any rooms after modification.

  5. the node type.(Type0: normal node. Type 1: floor entry node. Type 2: building entry node\textsuperscript{19}.)
  6. the node coordinates x,y

● Vector Class. It includes:
  1. the vector ID (automatically generated by the system)
  2. the node ID of the start node (the from-node)
  3. the node ID of the end node (the to-node)
  4. the length of the vector (the distance between start node and end node)

● Person Class. It includes:

\textsuperscript{19} For detailed information, see 6.1.4.
1. the person name (cannot be empty, given by user)
2. the floor ID (in which floor the person is located, if not assigned its value is -1)
3. the room ID (in which room the person is located, if not assigned its value is -1)

The person class design is similar to the node design, after user changes the room-design the system will automatically check the person's position according to its coordinates. If the person is not located in any room, then this person is set to unassigned.

4. the location of the person's image file. If not assigned then use the default image from the system.
5. the person's coordinate x,y. If not assigned they are (0,0).

### 7.2 Main Window Functions

Main Window implements Ribbon buttons\(^20\) to execute the following tasks:

- **Add Building**

  The system uses „\My Documents\XGraphicProjects“ folder as the default working directory. When a new project is created, the system first asks the user to give a name to the new building. Each building must have an unique name.

  After the user gives a valid name to the new building, a sub-folder with the new project name will be created in the working directory. The system pops up the following message box to inform the user. Meanwhile the corresponding building instance is created in the system as well.

\(^20\) For detailed explanation, see 4.5.
\(^21\) The editor will get the current „\My Documents“ physical folder by code.
Delete Building

To delete a building the user must select the corresponding building item in the treeview. This is to ensure that the user doesn't delete the building incidentally. After the user's confirmation the system will delete all related user controls from all dictionaries, delete the corresponding building instance and delete its sub-folder in the working directory.

Check Building Graph

Although the indoor navigation system editor doesn't need to query the shortest-path between two nodes, with the help of „Check Building Graph“ function the user can check if the graph-design is valid.

The rules of graph-validity are:

1. In a floor every node can reach at least one floor entry node or one building entry node of this floor.
2. In a building every node can reach at least one building entry node.

Rule 1 will be checked when the graph-design user control is set from editing mode to the view mode\(^2\). By clicking „Check Building Graph“ the graph-validity of selected building item in the treeview will be checked by Rule 2.

Before the graph-validity check of Rule 2 begins, the system makes sure that all graph-design user controls of this building are in view mode. If a graph-design is in editing mode, the system can turn it into view mode by saving the modifications in it with the user's confirmation. If this graph-design user control does not satisfy the graph validity check Rule 1 so that can not be saved, the „check building graph“ process will be terminated and the corresponding graph-design will be displayed in the mainview. Only a building with all successfully saved graph-designs for each floor can execute the building graph's validity check.

The calculation of reachability for each node uses Dijkstra Algorithm\(^3\).

The complete algorithm is:

1. boolean reachable ← true
2. if no building entry node defined in the building
3. reachable ← false
4. if reachable == true
5. for each normal node Ni in the building
6. calculate the distance \(D_{ix}\) from \(N_i\) to every node \(N_x\) in the graph with Dijkstra algorithm.
7. for each building entry node \(N_j\) in the building
8. if distance \(D_{ij}\) == infinite
9. reachable ← false
10. break

---

\(^2\) For detailed explanation, see 7.5.
\(^3\) For detailed explanation, see 5.2.
11. if reachable == false
12. break
13. return reachable

If after checking building graph Rule 2 is not fulfilled, there may be two reasons:

1. There is no building entry node defined in this building. In this case the system will pop up a message box with the information „Graph design error! The current building must have at least 1 Building Entry Node. Pls. check your graph again!“

2. There is at least one normal node which can not reach any building entry node. In this case the system will pop up a message to inform the user about the problem node and highlight it with red color and display its corresponding graph-design user control in the mainview.

Taking into account that a building may have many floors, it is permitted by the system if the user wants to save the building even if its graph-design does not fulfill Rule 2. But for each floor's graph-design, the system will check it with Rule 1 before saving. In contrast to the method of storage of the building graph, a floor's graph-design which does not fulfill Rule 1 can not be saved (that means, this user control can not be set back into view mode), the system will highlight the problem node and ask the user to check it again. Only a Rule 1 fulfilling floor's graph-design can be saved into the building.

Drawing 22: The highlighted problem node

- Save Building
  
  From a macro perspective “save building” has two meanings:
1. Save changes into building instance. This will be done when a user control is set from editing mode back into view mode.

2. Save changes of a building instance into a xml file so that the result can be used by indoor navigation system.

With „Save building“ button click the 2. task will be accomplished. In this section when „save building“ is used, it refers to the 2. task.

By saving building the system first makes sure that all related room-design and graph-design user controls are in view mode. If a room-design is in editing mode, the system can turn it into view mode by saving the modifications in it with the user's confirmation. The process how to set an editing graph-design back to view mode was described in the „Check Building Graph“ section.

If a graph-design user control can not be saved due to the graph validity check “Rule 1 is not fulfilled”, the save building process will be terminated and the corresponding graph-design will be displayed in the mainview. This is similar to „Check Building Graph“.

After the „Save Building“ function is executed a Building.xml file is created according to the modified building instance and is saved into its sub-folder under the working directory.

- Add Floor/2D Picture

This function can add a 2D floor layout picture to the selected building. By clicking this button a sub-window is displayed, in which the computer file-structure is shown as a treeview. But in this treeview only the files with the extensions ".bmp .gif .jpeg .png .tiff .jpg .ico" will be shown.
After a user selected a 2D picture file and gave a valid and unique floor name for this 2D picture, the selected 2D picture file will be copied into the building sub-folder and renamed with the floor name. In the treeview the related floor item and its graph item are created. The program uses the FloorID to represent the height of the floor. So the new created floor is the highest floor in the building. When the user selects the floor item, the new added 2D picture file is shown as floor layout.

All added files or deleted files will be marked by the system. Only when the user saves all changes before he exits the program, all these files will be created or deleted indeed. Otherwise the system will ignore these changes.

● Change Picture

With this function the layout file of an existing floor can be changed.

First the system must make sure that all user controls of this building are in view mode. If the rooms and graph of this floor are already designed, the system can delete or keep them
according to the user's choice.

By deleting the rooms all persons which are assigned to this floor will be set to unassigned. By deleting the graph only the nodes in this floor and the vectors which are related to this floor will be deleted. This is to minimize the influence to the whole graph-design by the deletion of one special floor.

The floor layout file change process is similar to „Add Floor/2D Picture”.

• Delete Floor

To keep the correct sorting of the floors, only the highest floor in the building can be deleted. The precondition to delete a floor is that no graph-design user control of this building is in editing mode.

If this precondition is fulfilled, the system will do the following tasks:

1. Delete this floor instance from the list of floor instances of the building.
2. The rest work is the same as in section „Change Picture” when the user wants to delete the old room-design and graph-design of one floor.

• Load 3D Binfile/ Floor Design

With this button the functions which are described in 6.1.1 and 6.1.2 are realized. First of all there must be a binary file which is converted from a 3D data model for importation. The user can select this binary file with the help of the computer treeview sub-window (Drawing 23). But this time only the .bin files will be shown for selection.

After selecting a binary file the 3D data user control will be displayed in a sub-window, in which the 3D data model is displayed. The user can select the different layers to see their visualization, move and zoom them and make a 2D „snapshot“ from them. Through this function a 3D model can be transferred to a 2D floor layout.

25 For detailed explanation, see 7.3.
26 For detailed explanation, see 6.1.2.
In "Load 3D Bin File sub-window" on the left top is a listbox, in which all layers from the binary file are displayed. In the display sub-window the 3D model will be displayed in top view. By default all 3D objects will be displayed. By selecting a layer in the listbox, the other unselected layers will be set to invisible, and the user can select more than one layer to construct a floor-layout. After selecting the right layers, the user can zoom and move the model by mouse wheel and left button.
If the graph is already designed in the 3D data model, the system can import it as well with the help of the checkbox „Import Lines as Graph“. By checking this checkbox the lines in the selected model will be imported as graph-design of this floor. By clicking the button „Create Floor with selected Layers“, an input message box will be shown in which the user can give a name for the new floor.

If the floor name is valid and unique, the system will do the following work by implementing the functions which are supplied from X3D 3D visualization libraries: it will make a snapshot of the displayed 3D model and save this .png file in the related building sub-folder as the new created floor's layout file. At the same time the 3D coordinates of all selected polygons and polylines will be converted into the corresponding 2D coordinates. Each polygon and polyline will be imported as a room design with the name „RoomX“, X is the serial number starting from 1. If checkbox „Import Lines as Graph“ is checked, similarly the system will get the corresponding 2D coordinates of all vertices in selected lines. Each vertex will be imported as a node, each line will be imported as a vector. The RoomID attribute of each node will be located by the „Point-In-Polygon algorithm“.

For detailed explanation, see 6.1.1.
For detailed explanation, see 4.6.
For detailed explanation, see 5.3.
The layers which were used to create a new floor will be disabled in the listbox. The user can repeat the aforementioned steps until all possible floors are created from the 3D data model.

By clicking button „Reset all“ all intermediate working results (imported floors, rooms, graph) will be cleared so that the user can restart his work again from the beginning.

By clicking button „Transfer floors definition to main window“ all the new created floor/room/node/vector instances will be transferred into building instance, the Load 3D Bin File sub-window will be closed and all the new created instances will be shown in the main window accordingly. The user can view and edit the imported floor design, room-design and graph-design as well as other 2D floors in the mainview.

![Drawing 27: The imported 2D room-design from 3D bin file](image-url)
7. Implementation details - Editor

- By clicking button „Cancel“ the load bin file sub-window will be closed and the system returns to the main window without doing anything.

- Add Person

  With this function the user can add a person to the corresponding building.

  After inputting the name the system will check its validity (without invalid characters\(^{30}\) and not empty) and check if it already exists in the building. The person's name must be unique in one building. If there are persons with the same name in one building, the user should add additional information (e.g. the department) in the person name so that they are different from each other.

  The user can add a personal photo for the created person immediately or later with the help of the „Add/Change Photo“ button. If the user wants to add the personal photo immediately, he can choose a picture file from the computer treeview sub-window (Drawing 23). This picture file will be copied into the building's sub-folder and renamed with the person's name. If the user doesn't want to add the personal photo immediately, the system will give a default image for the person.

  A person instance which location is unassigned will be created accordingly.

- Add/change Photo

  With this button changing or adding of a person's photo afterwards is possible.

\(^{30}\) Invalid characters are: /, \. * ? , " < > . These characters are invalid for creating a file name in Windows.
7. Implementation details - Editor

- **Delete Person**
  
  By clicking button „Delete Person“ the selected person will be deleted from the building after user's confirmation. If this person's location is already assigned to a floor, then refresh this floor's room-design user control if necessary.

- **Exit**
  
  Before closing the editor the system will make sure that all user controls are in view mode. After that if there are some unsaved changes the system will ask the user whether he wants to save the changes or not. By answer „Yes“ for each changed building the function „Save Building“ will be executed. By answer „No“ the system will restore the marked files with the opposite action: delete the new added files and restore the deleted files.

### 7.3 The precondition for main window's functions

After the explanation in 6.2.2 and 7.2, now we can use a table to list the preconditions for the main window's important functions. All these preconditions will be checked by the system automatically and it will ask the user whether he wants to save the changes into the corresponding building instance by setting the editing user control back to view mode.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Preconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Building</td>
<td>1. No graph-design of this building is in editing mode.</td>
</tr>
<tr>
<td></td>
<td>2. No room-design of this building is in editing mode.</td>
</tr>
<tr>
<td>Change Picture</td>
<td>1. This floor's room-design is not in editing mode.</td>
</tr>
<tr>
<td></td>
<td>2. No graph-design of this building is in editing mode.</td>
</tr>
<tr>
<td>Set room-design user control to editing mode</td>
<td>No graph-design of this building is in editing mode.</td>
</tr>
<tr>
<td>Set graph-design user control to editing mode</td>
<td>1. No graph-design of this building is in editing mode.</td>
</tr>
<tr>
<td></td>
<td>2. No room-design of this building is in editing mode.</td>
</tr>
<tr>
<td>Delete Floor</td>
<td>No graph-design of this building is in editing mode.</td>
</tr>
<tr>
<td>Exit</td>
<td>1. No graph-design of all buildings is in editing mode.</td>
</tr>
<tr>
<td></td>
<td>2. No room-design of all buildings is in editing mode.</td>
</tr>
</tbody>
</table>

*Table 2: The precondition for main window functions*

### 7.4 Room editor

The room editor is the room-design user control in editing mode. In room editor the user can:

---

31 In sake of more efficiency, the system will ask user to save the changes only if there are some changes not saved at the time point when user wants to exit the program.

32 For detailed explanation, see previous „Save Building“ section in this chapter.

33 For detailed explanation, see previous „Add Floor/2D Picture“ section in this chapter.
1. Create new room by mouse clicks.

Each valid left button mouse click will create a new point in the floor and the system saves this point into a points buffer. A „valid“ left button mouse clicks means that it is not located in any existing polygons, this will be checked with the „Point-in-Polygon“ Algorithm (see 5.3). Each right mouse click will create a new room with all points in the points buffer and clear the buffer afterwards. Each room will be represented in the program as a WPF polygon. A WPF polygon can be a complex polygon. In computer graphics, a complex polygon is a polygon which intersects itself. A complex polygon is not a valid room. In this program the system will not check if the polygon is simple or not by creating a new room. The user must ensure that the sequence to create new points is consistent in the sequence of a simple WPF polygon. e.g. If the points sequence (A,B,C,D) builds up a simple WPF polygon, then the sequence (A,C,B,D) builds up a complex polygon. The user can delete the invalid rooms. This is very easy to achieve by the eyes observation.

A new room can not be created if there are less than 3 points in the buffer.

Automatic detection of a complex polygon is possible, but the system can not determine the correct points sequence that the user wants to have.

E.g. as shown in Drawing 31, a complex polygon ABCDEFGH is created by the user. The system detects that this polygon is complex but it can not determine which simple polygon is the correct one. Since in this case there are at least two resonable rooms ABDCEFHG and ABDFHGECD.

Because of this the system does not make any automatic complex polygon dection and correction.
2. Use existing points to create a new room.

By right clicking on a selected point, a popup menu enables the user to use this point to create a new room. If the user clicks on this menu item, a new point will be created at the same position of the selected point and added into the points buffer. In this way after all necessary new points are created either from the existing points or not, if the creating sequence is correct, an adjacent room can be easily created.

Although these two overlapping points make the two adjacent rooms intersecting (they have common boundary or touch at a point), but this is permitted in room-design. The program uses an improved algorithm which is based on “polygon collision problem” and treats these two rooms as a valid room-design.

3. Add/Delete points to/from an existing room.

The user can select a room or a point by left clicking in the corresponding room or the point. The selected point will be highlighted in red color. For a selected room its edges will be shown in red color and its points will be shown in yellow color except one point which information is shown in the floor property window will be highlighted in green color. For a selected point the room to which it belongs will be highlighted in yellow color.

---

34 For detailed explanation, see 5.4
35 For detailed explanation, see 8.1.
User can delete the selected room or point from its room by pressing the „Delete“ key. By deleting a point in a room that has only 3 points, the system will pop up a message to inform user that he should select the whole room to delete.

By right click on a selected room the system will show a popup menu to enable the user to add points to this polygon.

By clicking on this popup menu the system will ask user to input the point IDs between which the new points will be added. User can get the point's ID from a popup context menu by putting the mouse on the point. Invalid point's IDs will not be accepted.

Since in WPF a complex polygon is permitted, the user must pay attention to the adding points' sequence so that it matches the from point and to point's direction. This difference is shown in Drawing 37 and Drawing 38.
4. Change the position of an existing room/ point.

User can select a room or a point and move it to the desired position by mouse drag-and-drop. In the property window the coordinates of selected point will be updated simultaneously. By releasing the mouse the system will check if the change of point's position causes polygon collision problem with the help of „Edge intersection test“ algorithm in 5.4. The point's new position which causes polygon collision is taken by the system as invalid and the point will be set back to the old position after the system shows an error message to the user.

5. Edit the room's name.

By creating a new room (polygon) the system will create a textbox in the middle of the polygon which text is „RoomX“, x is the increasing serial number beginning with 1. User can edit the room's name in the textbox, or use the default name „RoomX“ as the room name.

6. Save the new room-design of a floor.

Save the new room-design of a floor happens when an in-editing-mode room-design user control is set back to view mode and the user wants to save the changes. All the changes in the current floor will be saved into building instance. This is the 1. task which is mentioned in 7.2 „Save Building“ section.

By saving the new room-design the system will do the following actions:

1. create a new list of rooms which is named as „rooms“.

2. convert all polygons in the room-design user control into room instances and add them into „rooms“. For each room instance the system will create a list of points according its
room-design. The roomIDs and pointIDs start from 1 to n.

3. Set “rooms” as the roomlist of the current floor.

4. If the graph of this floor is already designed, the system must relocate the nodes in the updated room-design. The working principle is using the coordinates of each node and the „Point-in-Polygon“ algorithm\(^{37}\) to get the current valid RoomID. If a node is located in none of the rooms, then this node is invalid and will be deleted from the building instance. Its related vectors will be deleted as well.

5. If in this building persons are already defined the system must relocate the persons in the updated room-design. The system first gets a sublist of persons whose floorID is the current floor from the building's definition and check if each person still exists in the current floor. If this person doesn't exist any more, this means that this person is relocated to another floor or deleted from this floor, then set this person's floorID and roomID to -1 as unassigned. For all persons which exist in the current floor, the system will get each person's roomID with the help of its coordinates and the „Point-in-Polygon“ algorithm.

The methods to define the person's position will be described in 7.6.

### 7.5 Graph editor

The graph editor is the graph-design user control in editing mode. In the graph editor the user can:

1. Create new nodes and new vectors per mouse click

   Each valid left button mouse click will create a new node in the floor and the system saves this node into a nodes buffer. A „valid“ left button mouse click means that it must be located in an existing room, this will be checked with „Point-in-Polygon“ Algorithm (see 5.3). Because each vector can only have two nodes, so that if there are already two nodes in the nodes buffer, the system will popup a message to inform the user that „There are already two nodes for creating a new vector.“

   Each right mouse click will create a new vector with the two nodes in the nodes buffer and clear the buffer afterwards. The user can also use an existing node to create a new vector by clicking the menu item „Use this node to create new vector“ in the contextmenu which will be displayed when a user right clicks a node. In the graph editor a vector is edited indirectly by editing its nodes. When a node is moved, the related vectors will be moved accordingly; when a node is deleted, the vectors which includes that node will be deleted as well.

---

\(^{37}\) For detailed explanation, see 5.3.
2. Edit node's name, type and its inter-floor-vectors

This can be done in two ways: in Graph Property Window or by right click on one node, the 1st Method will be explained in 8.2, in this section only the 2nd. method will be explained.

When user right clicks one node, a contextmenu will be displayed:

With the help of this contextmenu the user can do the following tasks:

• Add/Change node name

Although the system uses the NodeID as key to access a node, it is meaningless for the user and it is better to give every node a proper name so that later in the indoor navigation system the shortest-path can be better described. After clicking the menu item „Add/Change NodeName for NodeX“ an input message box will be displayed for inputting the new node name.

• Use the existing node to create new vector

By clicking the menu item „Use this node to create new vector“ the selected node will be added into the nodes buffer which can be used to create a new vector.

• Set the node as Entry Node
If the menu item „Set this node as EntryNode“ is checked, the selected node will be set as an Floor Entry Node.

- Set the node as Building Entry Node

If the menu item „Set this node as BuildingEntryNode“ is checked, the selected node will be set as Building Entry Node.

Once the node is set as a floor/building entry node, the last three menu items in the context menu will be enabled.

- Create inter-floor-vector for an entry node

As an Entry Node it can create inter-floor-vectors with other entry nodes from other floors. The menu items „Link this Node to EntryNode of other floor“ and „Link this Node to BuildingEntryNode of other floor“ can be used to do this work. If user clicks one of these menu items, a combobox will be shown, in which all Floor/Building Entry Nodes from other floors are listed. Each entry node is shown in „Floor Name | Node ID | Node Name“ format. By selecting one item an inter-floor-vector between the selected node and node listed in the item will be created.

![Drawing 41: All Floor Entry Nodes of other floors](image)

- delete inter-floor-vector for an entry node

With the help of menu item „Delete the inter-floor vector which includes this Node“ user can delete existing inter-floor-vectors which are linked with the selected Node. By clicking this menu item all inter-floor-vectors of this node will be shown in a combobox, by selecting one item this inter-floor-vector will be deleted.

![Drawing 42: All inter-floor-vectors with the selected node](image)

---

38 Die definition of Floor Entry Node is explained in 6.1.4
39 Floor Entry Node or Building Entry Node.
3. Edit node's position per mouse drag-and-drop

Like in the Room Editor in the Graph Editor the user can change the node's position with mouse drag-and-drop. During drag-and-drop the coordinates of the selected node will be changed accordingly in the graph property window as well as all vectors which are linked to this moving node. When the user releases the mouse, the system will use the "Point-In-Polygon" Algorithm\(^{40}\) to determine in which room the node is located and update the RoomID attribute of the moved node. If the node is located outside of all rooms, the system will popup an error message and set the node back to the old position.

4. Save the graph-design of this floor

In the initialization of the graph-design user control, the system will load all nodes of the current floor to create a nodeDictionary, which is also used for saving new created nodes during editing. The system also loads all vectors which include the nodes of the current floor and save them into lineDictionary, which is also used for saving new created vectors during editing.

Saving the new graph-design of a floor happens when an in-editing-mode graph-design user control is set back to view mode and the user wants to save the changes. By saving the new graph-design all the changes in the current floor will be saved into building instance. Because the graph is saved in the whole building not in each floor, the storage of the graph-design of one floor will affect all graph-designs of other floors.\(^{41}\)

By saving the new graph-design of a floor into the building instance the system will do the following steps:

1. convert all KeyValuePairs in nodeDictionary to a list of nodes (cnodes).
2. for each KeyValuePair in lineDictionary
   2.1 convert it into vector instance
   2.2 calculate the distance of each vector by using the formula in 6.1.4.
   2.3 add the vector to one of three different vector lists cvectors, fromcvectors, tocvectors according to the next 3 conditions. These 3 lists will be used for later nodeID changes (in cvectors the from-node and to-node IDs will be both changed, in fromcvectors only the from-node ID will be changed, in tocvectors only the to-node ID will be changed).
      2.3.1 if the from-node and to-node both belong to the current floor, then add this vector to list cvectors.
      2.3.2 if only the from-node belongs to the current floor, then add this vector to list fromcvectors.
      2.3.3 if only the to-node belongs to the current floor, then add this vector to list tocvectors.
3. check the new graph-design validity of the current floor with Dijkstra Algorithm\(^{42}\), the checking rule is the Rule 1 which is explained in 7.2 "Check Building Graph" section:

\(^{40}\) For detailed explanation, see 5.3
\(^{41}\) For detailed explanation, see 6.4
\(^{42}\) For detailed explanation, see 5.2
3.1 For each vector in \texttt{cvectors}, create a reverse vector and add it into a vector list \texttt{reversedcvectors};

3.2 vector list \texttt{dijkstracvectors} ← \texttt{cvectors} + \texttt{reversedcvectors};

3.3 create a Dijkstra instance with \texttt{dijkstracvectors} and \texttt{cnodes};

3.4 \texttt{Dijkstra} \texttt{d} = \texttt{new Dijkstra(dijkstracvectors, cnodes)};

3.5 if there is no entry node in this floor defined, the system will popup an error message and terminate the current function;

3.6 check if each normal node in \texttt{cnodes} can reach at least one floor entry node or building entry node in this floor. If not, then highlight the error node, popup an error message and terminate the current function;

4. if the dijkstra's check of new graph-design is OK, then do the following steps to insert the new designed nodes and vectors into the building instance:

4.1 assign all old designed nodes from building instance into the node list \texttt{oldnodes};

4.2 remove all nodes of this floor from \texttt{oldnodes};

4.3 assign all old designed vectors from building instance into the vector list \texttt{oldvectors};

4.4 remove all vectors which include the nodes of this floor from \texttt{oldvectors};

4.5 get the maximum NodeID \texttt{maxindexincnodes} from \texttt{cnodes};

4.6 sort \texttt{cnodes} ascendingly and change each NodeID to \texttt{maxindexincnodes + n}, \texttt{n} = 1, 2, \ldots, \texttt{n}. This change must be done in a descending loop (The change must be done from the highest NodeID in \texttt{cnodes})

4.7 change the NodeID in \texttt{cvectors}, from \texttt{cvectors}, to \texttt{cvectors} accordingly;

4.8 change each NodeID in \texttt{cnodes} to \texttt{n}, \texttt{n}=1, 2, \ldots, \texttt{n}. This change must be done in an ascending loop (The change must be done from the lowest NodeID in \texttt{cnodes});

4.9 change the NodeIDs in \texttt{cvectors}, from \texttt{cvectors}, to \texttt{cvectors} accordingly;

4.10 get the maximum NodeID \texttt{cnodestartindex} from \texttt{oldnodes};

4.11 sort \texttt{cnodes} ascendingly and change each NodeID to \texttt{cnodestartindex + n}, \texttt{n} = 1, 2, \ldots, \texttt{n}. This change must be done in a descending loop

4.12 change the NodeID in \texttt{cvectors}, from \texttt{cvectors}, to \texttt{cvectors} accordingly;

4.13 add \texttt{cnodes} into \texttt{oldnodes};

4.14 add \texttt{cvectors}, from \texttt{cvectors}, to \texttt{cvectors} into \texttt{oldvectors};

---

43 For detailed explanation, see 6.1.4 „Vector design“ section

44 The goal of this step is to change all NodeIDs in \texttt{cnodes} into a continuous set of positive integers, the discontinuities of NodeIDs which are caused by user adds or deletes created nodes should not affect the final result of nodes definition.

45 The goal of this step is to change all NodeIDs in \texttt{cnodes} so that they don't have conflict with the exiting NodeIDs from other floors, finally all nodes from all floors are saved together under the graph instance in the building instance.
4.15 sort oldnodes ascendingly and change each NodeID to \( n, n=1,2,...n \);

4.16 change the NodeIDs in oldvectors accordingly;

4.17 sort oldvectors ascendingly and change all VectorIDs in oldvectors to \( n, n=1,2,...n \);

4.18 set the oldnodes and oldvectors as the new graph-design of the building instance.

After this the new graph-design of this building has a list of nodes and a list of vectors which both IDs constitute a continuous set of positive integers starting from 1.

As we have seen the IDs of nodes/vectors of other floors can be changed after the user saved the modified graph-design of one floor. That's the reason why in one building at the same time only one floor's graph-design is permitted to be set into editing mode. After the description of graph-design saving we can now better understand 7.3 „The precondition for main window's functions“.

### 7.6 Person editor

The user can edit one person's position in two methods:

1. per mouse drag and drop into the floor which is in editing mode.

2. Allocate the person into one room of the building in a person's property window.

In this section only Method 1 will be explained in details. Method 2 will be introduced in 8.3.

If a room-design user control is in editing mode, the user can drag the person's icon from the treeview sub-window and drop it into the expected position in the floor. By mouse release the system will delete this person from other loaded room-design user control if it is in one of them.

After one person is dropped into the editing floor, it will be added into a person dictionary.

The user can select this person and change its position in the floor per mouse drag and drop, in the

*Drawing 43: Person Editor per mouse drag and drop*
person's property window the coordinates of this person will be changed accordingly.

If user selects this person and then presses the „Delete“ key, this person's icon will be deleted from the floor and the person's location is set as unassigned. At the same time in the person dictionary this person's item will be removed.

The modified position of persons will be saved when the modified room-design of this floor is saved. This is explained in details in 7.4 „Save the new room-design of a floor“ section Point 6.
8 Property user controls design

8.1 Floor Properties user control

If in the treeview sub-window floor/rooms/room items are selected, in the property sub-window the Floor Properties user control will be displayed. The content displayed in the Properties user control is different according to the selected treeview item.

1. By floor item

   If a floor item in the treeview is selected, in the main window the corresponding floor-layout picture will be displayed. In the floor properties sub-window, the following properties are displayed:
   • Floor name
   • Floor Plan Filepath
   • Number of rooms

   All these properties are read-only because the user can only view the floor layout picture. The selected room name and selected point are null.

2. By “Rooms” folder item

   If a “Rooms” folder item is selected, in the main view the corresponding room-design user control will be displayed.

*Drawing 44: The floor item's property window*
If this floor is in view mode, the properties sub-window displays the same contents as by its floor item.

If this floor is in editing mode, the properties sub-window displays the current information of the number of rooms, the selected room or point.

In Drawing 45, the room Wohnzimmer is selected, its information is shown in the properties sub-window. User can use the minus or plus button besides „Included Points“ to see the information of each point which belongs to the selected room. The number shown between the two buttons is the PointID. In the main view the point which information in shown in the „Included Points“ will be highlighted in green color. If the selected room is being moved per mouse drag-and-drop, the textboxes „X“ and „Y“ under „Included Points“ will show the real-time coordinates of the green point.
If a point is selected, the information of the selected point is shown in the 3rd frame in the properties sub-window. These information are:

- selected point ID
- the room name to which the selected point belongs
- its coordinates

In the main view, the selected point will be highlighted in red color. If user moves the selected point per mouse drag-and-drop, the textbox „X“ and „Y“ under „Selected Point“ will show the real-time coordinates of the selected point.

3. By room item

If a room item in the treeview sub-window is selected, in the main view the corresponding room-design user control is displayed in which the selected room is this room item. The displayed information in the property window is as same as when a room is selected in an in-editing-mode room-design user control. The only difference is here the user can only view it, not edit it.

When a room-design user control is set into editing mode, all the room-items in this floor will be hidden in the treeview sub-window, at the same time only the „Rooms“ folder of this floor is visible and changed to editing icon. This is because during editing rooms can be added, changed or deleted, it costs unnecessary time and codes if for each small change the user control has to trigger an event to the main window so that in the tree view the proper room-items can be displayed. Only when the room-design user control is set back to view mode, the system will refresh its room-items according to the final room-design.
Drawing 47: The difference between in editing and view mode floors in the treeview
8. Property user controls design

8.2 Graph Properties user control

When a graph item (to be exact a floor item in „Graphs“ folder) is selected in the tree view, the Graph Properties user control will be displayed in the property sub-window.

1. When the graph-design user control is in view mode, the user can not edit graph of a floor in view mode, but he can view all nodes' information in the graph properties sub-window.

In the Graph properties sub-window the following information will be displayed:

- Floor name
- Selected node information

User can press „minus“ or „plus“ buttons to change the selected Node ID, and in the main view sub-window the corresponding node will be highlighted in red color. For a selected node, the following information will be displayed:

a. Node name
b. its coordinates
c. if it is a floor entry node, the checkbox „Floor Entry Node“ will be checked\textsuperscript{46}

d. if it is a building entry node, the checkbox „Building Entry Node“ will be checked

The two checkboxes and the last three comboboxes are disabled. These will be used for editing a node when the graph-design user control is in editing mode.

2. When a graph-design user control is in editing mode, the user can use the graph properties window to modify the selected node. These modifications can also be achieved by using the contextmenu which is popped up when user right clicks a selected node\textsuperscript{47}. When a node is selected, its information will be displayed in graph properties window. These information are:

- **Node Name**
  
  User can edit the selected node name via the Node Name textbox.

- **Its coordinates**
  
  The coordinates of the selected node will be real time updated when user changes the position of this node per mouse drag-and-drop.

  - **If it is a floor entry node**

    User can modify the selected node's type with this checkbox.

  - **If it is a building entry node**

    User can modify the selected node's type with this checkbox as well.

  - **Editing the inter-floor-vectors of the selected node if it is an entry node**

    If the selected node is a floor/building entry node, the last 3 comboboxes are enabled so that user can create/delete inter-floor-vectors of this node.

    a. Link this node to Floor Entry Node of other floor.

    In this combobox all floor entry nodes of other floors are listed. By selecting an item an inter-floor-vector between the selected node and this entry node is created.

    b. Link this node to Building Entry Node of other floor.

    The function is similar as a except in this combobox all building entry nodes of other floors are listed.

\textsuperscript{46} For detailed explanation of entry node, see 6.1.4 and 7.5.

\textsuperscript{47} For detailed information, see 7.5 Point 2: Edit node's name, type and its inter-floor-vectors.
c. Delete the inter floor vector which includes this node.

In this combobox all inter-floor-vectors of the selected node are listed. By selecting an item means the inter-floor-vector between the selected node and this entry node will be deleted.

8.3 **Person Properties user control**

When a person item in the tree view is selected, the Person Properties user control will be shown in the property sub-window. If the location of the selected person is unassigned, in the main view no user control will be displayed, in the person properties window the following information will be displayed:

- **Person's name**
  
  User can change the person's name by typing a new name in the textbox, by pressing „Apply“ button or „Enter“ key the system will check the validity of the new name. If the new name is duplicated to another existing person's name, the system will pop up an error message and the text in the name textbox will be changed back to the old person name. If the new person name is valid, the person item in the tree view sub-window will be updated to the new name as well.

- **Person's coordinates**

- **Locate the selected person into a room**
  
  If the selected person is not assigned to any room the selected item of the combobox „Located in Floor/Room“ is null. But the user can use this combobox to assign a room to the person.

  By clicking the combobox all rooms in the building will be listed as its items. These rooms include the rooms which are just created and still not be saved into the building instance (Drawing 52).

  By selecting a room item means that the person is located to this room no matter its room-design user control is in editing mode or not. The person's coordinates will be updated in the following steps:

  1. Calculate the central point of the located room with its coordinates;

48 The operations are similar to 7.5 Point 2: Edit node's name, type and its inter-floor-vectors
2. Check if this central point is located inside the room with the help of „Point – in-Polygon“ Algorithm which is described in 5.3. If yes, then update the person's instance with the coordinates of center point and terminate the procedure.

3. If no, for each point of the located room:
   a. get the point's coordinates (x,y);
   b. check if \((x+\Delta, y+\Delta)\) is located inside the room; this step will be repeated until the check result is true;
   c. update the person's instance with the coordinates of point \((x+\Delta, y+\Delta)\).

Step 3 will be used when a room has irregular shape that its central point is not located inside it.

After locating the person into a room from its properties window in the main view the corresponding room-design user control will be shown with the highlighted person icon as the selected element. If this person was assigned to another floor, the system will automatically delete its image from the old floor.

49 In the program \(\Delta\) is currently set to 3.
If the selected person is located into a new room which is still not saved into the building instance, and the user leaves the editing mode without saving the changes, the system will load the old room-design into the user control, set the persons whose position does not locate in any rooms as unassigned and delete their images from the user control.

The user can delete person from the assigned room even if the corresponding room-design user control is not in editing mode. By selecting the person from the treeview, if this person's position is already assigned, the corresponding room-design user control will be displayed in the main view with that person as the selected element. If the user presses “Delete” key and at the same time the mouse is not focused in property window (If the user selects the person's name in property window and press Delete key, only the person's name not the person should be deleted), the selected person will be set to unassigned.

If the selected person is assigned to a room which user control is in editing mode, user can change this person's position again with mouse drag-and-drop.

In 7.6 it is already mentioned that there are two methods to edit a person's position:

1. per mouse drag and drop into the floor in editing mode.
2. Allocate the person into one room in the building in person's property window.

The difference between Method 1 and 2 is: With Method 1 the person's position will be updated when the modified room-design is saved, the intermediate states will be saved into the person dictionary; with Method 2 the person's position will be updated immediately into the building instance. That is because when a person is assigned from property window, the target floor may be not in editing mode therefore we can not use its saving procedure to

---

50 In 7.6 the usage of persons dictionary is explained.
8.4 Switch between floor- and person-property windows

For a floor in editing mode, the property sub-window can be changed accordingly when the selected element is changed. If the selected element is a room or a point, the floor properties will be displayed; if the selected element is changed to a person, the person properties will be displayed. At the same time the selected item in the treeview will be also switched from the „Rooms“ folder item to the selected person item. And vice versa.

9 Conclusions

9.1 Statement of the result

In the current version of the indoor navigation system the preparation of geometry- and building data is currently offered as a service from XGraphic engineering company and requires partly time-consuming processing according to the quality of the available data. But it happens quite often that the building is rearranged or extended so that the floors and rooms could be partly changed as well. It takes extra time for the XGraphic engineering company to modify the data model and it makes extra costs for the customers. The indoor navigation system editor simplified this work, so that the customer can establish and maintain the geometry- and building data without specific IT knowledge.

The aim of this master thesis is the design and the implementation of a geometry and data editor for graphical editing of the models and configurations of the data contents in the indoor navigation system.

After editing the room-design and graph-design of a building, the result is saved in the file Building.xml which content is listed in 10. „Annex – Building.xml“. This xml file can be used by the indoor navigation system to construct 2D data model in which the user can query the shortest path to persons, rooms or events by their names.

9.2 Open issues

9.2.1 GUI refinement

9.2.1.1 More friendly user experience

The program is only a prototype which can be improved with more functions that make the user feel more comfortable and friendly. These functions can be:

1. The copy/paste function.
   For a big building, it can happen that many rooms have the same shape. It is very boring and inefficient if the user has to create all these rooms with mouse clicks. The system can have a „Copy“ button, which can copy the selected room and add the copy into the floor by clicking „Paste“ button.

2. The undo/restore function.
With „undo“ function the system can set the status of the room-design or graph-design back to the status before the last action started. With „restore“ function the system can set it to the status after the last action finished. For that the system must store the last 2 status of the in-editing-mode user control.

3. Simplification of editing

From the user's perspective switching between the view and editing mode is not very convenient. All the room-design and graph-design user controls can always be in editing mode if all the validations will be done when „Save Building“ button is clicked. In case of design errors, an error list can be created and presented to the user.

After programing the core functions (room-design and graph-design) the easiest way to implement these two user controls in the main window is to use the view/editing mode switch. The idea of the simplification came up during the writing of this thesis. A change of the complete design idea would exceed the prescribed time-limit of a master thesis.

### 9.2.1.2 Menu optimization

The GUI design can be improved with more professional and esthetic icons. The sub-windows design can be more embellished and optimized. All the Ribbon buttons can be grouped into several sub-groups so that a clearer relationship can be established among the various functions.

### 9.2.1.3 Multilanguage support

In the current version the working language is hard coded in English. The system can implement WPF ResourceDictionary to realize the multilanguage support so that user can switch the working language between e.g. English and German.

### 9.2.2 Improvement of the functions in properties user controls

Due to the limited working time of a master thesis, in current version the properties user controls are mainly used to display the information of selected element in the main view. The following improvements could be implemented to enhance the user experience.

- **In Floor Properties user control**
  1. Set the „Floorname“ and „Roomname“ textboxes to be enabled so that the user can change the floor name and room name from property window.
  2. In editing mode, if a room or a point is selected, the user can edit the coordinates in „X“ and „Y“ textboxes to change the related point's position.

- **In Graph Properties user control**
  1. Set the „Floorname“ textbox to be enabled so that the user can change the floor name in it.
  2. In editing mode, if a node is selected, user can edit the coordinates in „X“ and „Y“ textboxes to change the related node's position.

- **In Person Properties user control**
  1. Add a button besides the „Imagepath“ textbox to enable user to change the person's
picture. This button should have the same function as „Add/Change Photo“ function in the main window.

2. User can edit the coordinates in „X“ and „Y“ textboxes to change the related person’s exact position.

9.3 **Outlook**

9.3.1 **Output file in 3D**

The output file of the current version of indoor navigation system editor can be used for 2D indoor navigation system, with some effort it can be expanded to 3D.

There are two ways to expand the program so that the output file can be used to construct a 3D data model.

1. Create a 3D data model based on the room-design.

   In this case the height difference between 2 adjacent floors will be set to be a certain value $\Delta z$. The lowest floor has the height 0. Each point/node’s 2D coordinates will be converted to 3D by adding a $z$ coordinate which is the height of the floor in which it is located. From each edge of a room a wall polygon will be created. The working principle is: Suppose an edge has 2 vertices $(x_1,y_1,z)$ and $(x_2,y_2,z)$, the corresponding wall polygon will be created with 4 vertices $(x_1,y_1,z)$, $(x_2,y_2,z)$, $(x_2,y_2,z+\Delta z)$, $(x_1,y_1,z+\Delta z)$.

   ![Drawing](attachment:image.png)

   *Drawing 55: Convert a 2D room to a 3D room*

   In this way each 2D floor can be converted to a 3D data model in which the graph-design is shown on the „bottom“ in it. If the shortest-path is over more than one floor, the user can use „Page up“ and „Page down“ keys to switch among the different floors.

2. Convert the 2D graph-design back to the original 3D building layout (only possible if the source file is in 3D).

   When the 3D data model is imported to create a new floor, the system needs to capture the „$z$“ coordinates of all 3D geometry objects in selected layers and calculate the average value „$Z_{center}$“ of them. At the time when snapshot is made, the system will record the current zooming and moving factors.

51 For detailed description, see 7.2 „Load 3D Binfile/Floor Design“ section.
When a building's room-design or graph-design is saved, for every point or node or person, its coordinates (x, y) can be converted back into 3D (x', y', Zcenter) with the help of:

a. the corresponding zooming and moving factors

b. ScreenToWorld() function from X3D.OCX

After this conversion the room-design and graph-design of this building are in 3D format and can be imported back into the original 3D data model.

9.3.2 Mobile client

As mentioned in 6.1.5, the program can be easily extended to edit the positions of equipments (e.g. servers, printers, infrastructures, firefighting points, Quick Response (QR) codes etc.) and activities (e.g. meetings, seminars, etc.) as well.

Suppose a customer company uses QR codes to identify different positions in a building. The positions of all QR codes can be edited in the indoor navigation system editor, and the final result file Building.xml can be transferred to the end user's smart phone via WLAN. A mobile client program can be developed with which the user can query the shortest-path from his position to another node. It works like this: the user scans a QR code which is located at his position, the program interprets the position of this QR code with the help of the file Building.xml as the start point. The user can select his destination from all other nodes defined in the graph. The program will calculate the shortest-path and show the path in the floor-layouts.
<?xml version="1.0" encoding="utf-8"?>
<XGraphic>
  <Building ID="1" Name="building1">
    <CFloor2Ds>
      <CFloor2D ID="1" Name="EG" FilePath="\XGraphicProjects\Building1\Erdgeschoss.png">
        <CRooms>
          <CRoom ID="1" Name="Wohnzimmer">
            <CPoint ID="1" X="19,545501065244" Y="50,04" />
            <CPoint ID="2" X="467,545501065244" Y="54,04" />
            <CPoint ID="3" X="453,545501065244" Y="382,04" />
            <CPoint ID="4" X="18,545501065244" Y="380,04" />
          </CRoom>
          <CRoom ID="2" Name="Küche">
            <CPoint ID="5" X="18,545501065244" Y="391,04" />
            <CPoint ID="6" X="94,545501065244" Y="390,04" />
            <CPoint ID="7" X="94,545501065244" Y="467,04" />
            <CPoint ID="8" X="258,545501065244" Y="470,04" />
            <CPoint ID="9" X="257,545501065244" Y="717,106913545941" />
            <CPoint ID="10" X="15,545501065244" Y="717,106913545941" />
          </CRoom>
          <CRoom ID="3" Name="Diele">
            <CPoint ID="11" X="378,545501065244" Y="391,106913545941" />
            <CPoint ID="12" X="453,545501065244" Y="391,106913545941" />
            <CPoint ID="13" X="453,545501065244" Y="599,106913545941" />
            <CPoint ID="14" X="264,545501065244" Y="601,106913545941" />
            <CPoint ID="15" X="267,545501065244" Y="469,106913545941" />
            <CPoint ID="16" X="378,545501065244" Y="470,106913545941" />
          </CRoom>
          <CRoom ID="4" Name="Gaeste-WC">
            <CPoint ID="17" X="266,545501065244" Y="614,106913545941" />
            <CPoint ID="18" X="378,545501065244" Y="614,106913545941" />
            <CPoint ID="19" X="369,545501065244" Y="715,106913545941" />
            <CPoint ID="20" X="265,545501065244" Y="716,106913545941" />
          </CRoom>
          <CRoom ID="5" Name="HAR">
            <CPoint ID="21" X="381,545501065244" Y="615,106913545941" />
            <CPoint ID="22" X="454,545501065244" Y="616,106913545941" />
            <CPoint ID="23" X="454,545501065244" Y="719,106913545941" />
            <CPoint ID="24" X="379,545501065244" Y="718,106913545941" />
          </CRoom>
          <CRoom ID="6" Name="Abstellraum">
            <CPoint ID="25" X="94,545501065244" Y="389,04" />
            <CPoint ID="26" X="378,545501065244" Y="390,106913545941" />
            <CPoint ID="27" X="378,545501065244" Y="469,106913545941" />
            <CPoint ID="28" X="94,545501065244" Y="466,04" />
          </CRoom>
        </CRooms>
      </CFloor2D>
    </CFloor2Ds>
  </Building>
</XGraphic>
<CRoom ID="2" Name="ElternSchlafZimmmer">
  <CPoint ID="5" X="237,545501065244" Y="29,04" />
  <CPoint ID="6" X="451,545501065244" Y="33,04" />
  <CPoint ID="7" X="452,545501065244" Y="361,04" />
  <CPoint ID="8" X="242,545501065244" Y="359,04" />
</CRoom>

<CRoom ID="3" Name="Flur">
  <CPoint ID="9" X="17,545501065244" Y="370,102168036069" />
  <CPoint ID="10" X="101,545501065244" Y="370,102168036069" />
  <CPoint ID="11" X="101,545501065244" Y="444,102168036069" />
  <CPoint ID="12" X="377,545501065244" Y="448,102168036069" />
  <CPoint ID="13" X="377,545501065244" Y="371,102168036069" />
  <CPoint ID="14" X="451,545501065244" Y="371,102168036069" />
  <CPoint ID="15" X="452,545501065244" Y="522,102168036069" />
  <CPoint ID="16" X="17,545501065244" Y="519,102168036069" />
</CRoom>

<CRoom ID="4" Name="HWR">
  <CPoint ID="17" X="18,545501065244" Y="531,102168036069" />
  <CPoint ID="18" X="201,545501065244" Y="531,102168036069" />
  <CPoint ID="19" X="200,545501065244" Y="696,102168036069" />
  <CPoint ID="20" X="16,545501065244" Y="694,102168036069" />
</CRoom>

<CRoom ID="5" Name="Badzimmmer">
  <CPoint ID="21" X="211,545501065244" Y="696,102168036069" />
  <CPoint ID="22" X="210,545501065244" Y="532,102168036069" />
  <CPoint ID="23" X="451,545501065244" Y="534,102168036069" />
  <CPoint ID="24" X="453,545501065244" Y="699,102168036069" />
</CRoom>
</CRooms>
</CFloor2D>

<CFloor2D ID="3" Name="DG" FilePath="\XGraphicProjects\Building1\Dachgeschoss.png">
  <CRooms>
    <CRoom ID="1" Name="Arbeitszimmer">
      <CPoint ID="1" X="13,545501065244" Y="35,04" />
      <CPoint ID="2" X="447,545501065244" Y="34,04" />
      <CPoint ID="3" X="448,545501065244" Y="353,04" />
      <CPoint ID="4" X="16,545501065244" Y="354,04" />
    </CRoom>
    <CRoom ID="2" Name="Flur Oben">
      <CPoint ID="5" X="17,545501065244" Y="441,960821350274" />
      <CPoint ID="6" X="101,545501065244" Y="444,960821350274" />
      <CPoint ID="7" X="101,545501065244" Y="432,960821350274" />
      <CPoint ID="8" X="17,545501065244" Y="432,960821350274" />
    </CRoom>
    <CRoom ID="3" Name="Spielzimmer">
      <CPoint ID="9" X="15,545501065244" Y="441,960821350274" />
      <CPoint ID="10" X="376,545501065244" Y="444,960821350274" />
      <CPoint ID="11" X="377,545501065244" Y="683,960821350274" />
      <CPoint ID="12" X="16,545501065244" Y="683,960821350274" />
    </CRoom>
    <CRoom ID="4" Name="Technikraum">
      <CPoint ID="13" X="376,545501065244" Y="441,960821350274" />
      <CPoint ID="14" X="449,545501065244" Y="444,960821350274" />
      <CPoint ID="15" X="450,545501065244" Y="683,960821350274" />
      <CPoint ID="16" X="377,545501065244" Y="683,960821350274" />
    </CRoom>
  </CRooms>
</CFloor2Ds>
<CGraph>
 <CNodes>
  <CNode ID="1" Name="Arbeitszimmer" FloorID="3" RoomID="1" Type="0" X="57,0910021304879"
 Y="301,104122069546" />
  <CNode ID="2" Name="FlurDG" FloorID="3" RoomID="2" Type="1" X="59,0910021304879"
 Y="384,104122069546" />
  <CNode ID="3" Name="Spielzimmer1" FloorID="3" RoomID="3" Type="0" X="61,0910021304879"
 Y="492,104122069546" />
  <CNode ID="4" Name="Spielzimmer2" FloorID="3" RoomID="3" Type="0" X="330,091002130488"
 Y="493,104122069546" />
  <CNode ID="5" Name="Technikraum" FloorID="3" RoomID="4" Type="0" X="414,091002130488"
 Y="492,104122069546" />
  <CNode ID="6" Name="Wohnzimmer1" FloorID="1" RoomID="1" Type="0" X="51,545501065244"
 Y="321,146588058902" />
  <CNode ID="7" Name="Küche2" FloorID="1" RoomID="2" Type="0" X="51,545501065244"
 Y="429,146588058902" />
  <CNode ID="8" Name="Abstellraum" FloorID="1" RoomID="6" Type="0" X="127,545501065244"
 Y="428,146588058902" />
  <CNode ID="9" Name="Küche1" FloorID="1" RoomID="2" Type="0" X="201,545501065244"
 Y="530,146588058902" />
  <CNode ID="10" Name="Diele1" FloorID="1" RoomID="3" Type="0" X="307,545501065244"
 Y="529,146588058902" />
  <CNode ID="11" Name="GästeWC" FloorID="1" RoomID="4" Type="0" X="307,545501065244"
 Y="647,146588058902" />
  <CNode ID="12" Name="Hauseingang" FloorID="1" RoomID="3" Type="2" X="414,545501065244"
 Y="464,146588058902" />
  <CNode ID="13" Name="HAR" FloorID="1" RoomID="5" Type="0" X="414,545501065244"
 Y="328,146588058902" />
  <CNode ID="14" Name="Wohnzimmer2" FloorID="1" RoomID="1" Type="0" X="412,545501065244"
 Y="415,545501065244" />
  <CNode ID="15" Name="FlurNachOG" FloorID="1" RoomID="3" Type="1" X="412,545501065244"
 Y="297,04" />
  <CNode ID="16" Name="KinderSchlafZimmer" FloorID="2" RoomID="1" Type="0" X="52,545501065244"
 Y="485,04" />
  <CNode ID="17" Name="ElternSchlafZimmer" FloorID="2" RoomID="2" Type="0" X="415,545501065244"
 Y="295,04" />
  <CNode ID="18" Name="FlurNachUnten" FloorID="2" RoomID="3" Type="1" X="54,545501065244"
 Y="409,04" />
  <CNode ID="19" Name="FlurNachOben" FloorID="2" RoomID="3" Type="1" X="415,545501065244"
 Y="409,04" />
  <CNode ID="20" Name="Flur2" FloorID="2" RoomID="3" Type="0" X="307,545501065244"
 Y="87,763192635358" />
  <CNode ID="21" Name="Badzimmer" FloorID="2" RoomID="5" Type="0" X="308,545501065244"
 Y="584,763192635358" />
  <CNode ID="22" Name="Flur1" FloorID="2" RoomID="3" Type="0" X="103,545501065244"
 Y="843,763192635358" />
  <CNode ID="23" Name="HWR" FloorID="2" RoomID="4" Type="0" X="103,545501065244"
 Y="575,763192635358" />
 </CNodes>
 <CVectors>
  <CVector ID="1" From="16" To="17" Distance="108,018516931126" />
  <CVector ID="2" From="6" To="7" Distance="108" />
  <CVector ID="3" From="15" To="17" Distance="358,683288202289" />
  <CVector ID="4" From="1" To="2" Distance="83,0240928887513" />
  <CVector ID="5" From="17" To="22" Distance="92,772401116955" />
  <CVector ID="6" From="7" To="8" Distance="76,0065786626395" />
  <CVector ID="7" From="7" To="9" Distance="180,83417818543" />
  <CVector ID="8" From="18" To="19" Distance="114" />
  <CVector ID="9" From="2" To="19" Distance="357,327032029471" />
 </CVectors>
</CGraph>
<CVector ID="10" From="9" To="10" Distance="106,004716876184" />
<CVector ID="11" From="20" To="19" Distance="133,646328265059" />
<CVector ID="12" From="10" To="11" Distance="118" />
<CVector ID="13" From="2" To="3" Distance="108,018516931126" />
<CVector ID="14" From="20" To="21" Distance="97,0051545022221" />
<CVector ID="15" From="10" To="12" Distance="107,004672795163" />
<CVector ID="16" From="22" To="20" Distance="204,03921191771" />
<CVector ID="17" From="22" To="23" Distance="91,9999999999999" />
<CVector ID="18" From="12" To="13" Distance="116" />
<CVector ID="19" From="15" To="12" Distance="103,01941564579" />
<CVector ID="20" From="4" To="3" Distance="269,001858729638" />
<CVector ID="21" From="14" To="15" Distance="99" />
<CVector ID="22" From="4" To="5" Distance="84,0059521700695" />
</CVectors>
</CGraph>

<CPersons>
    <CPerson Name="Maya Engelbrecht" FloorID="1" RoomID="1" Image="\XGraphicProjects\building1\Maya Engelbrecht.png" X="238,045501065244" Y="216,04" />
    <CPerson Name="Markus Engelbrecht" FloorID="1" RoomID="3" Image="\XGraphicProjects\building1\Markus Engelbrecht.png" X="359,045501065244" Y="496,106913545941" />
    <CPerson Name="Person3" FloorID="-1" RoomID="-1" Image="" X="0" Y="0" />
</CPersons>
</CBuilding>
</XGraphic>
Bibliography

• DA00: Dr.-Ing. David Buttgereit, Stationäre und mobile Wegeleit- und Informationssysteme, 2012
• DR00: Darel Rex Finley, Point-In-Polygon Algorithm — Determining Whether A Point Is Inside A Complex Polygon, 2007, http://alienryderflex.com/polygon/
• GPWiKi00: Game Programming Wiki, Polygon Collision, 2012, http://content.gpwiki.org/index.php/Polygon_Collision