VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# NÁVRHÁŘ A GENERÁTOR BUDOV

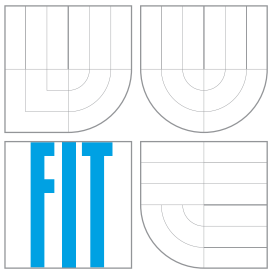
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK JURKA

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# NÁVRHÁŘ A GENERÁTOR BUDOV

BUILDING DESIGNER AND GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZDENĚK JURKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PŘIBYL JAROSLAV

BRNO 2010

# Abstrakt

Tato práce se zabývá návrhem a tvorbou 2D editoru pro rychlý převod bitmapové předlohy půdorysu budovy do vektorové reprezentace. Důraz je kladen na ucelený objektový návrh editoru umožňující jeho rozšíření o část automaticky generující 3D objektovou reprezentaci ze získaného vektorového půdorysu. V práci je popsána důležitá část týkající se návrhu uživatelského rozhraní a také neméně důležité matematické pozadí jednotlivých implementovaných funkcí 2D editoru. Práce se rovněž věnuje popisu zamýšleného generování 3D modelů s možností volby vzhledu výsledného 3D modelu.



# Abstract

This work deals with design and creation of 2D editor for fast transfer of bitmap building plans to a vector representation. Emphasis is placed on an object oriented design of editor allowing extension of the automatically generating 3D object representation obtained from the vector form. The thesis describes an important part of the design of the user interface and an equally important mathematical background of the implemented 2D features. The work also addresses the description of the intended generation of 3D models with a choice of appearance of the resulting 3D model.



# Klíčová slova

Editor, .NET, C#, plány budov, vektorová grafika

# Keywords

Editor, .NET, C#, ,building plans, vector graphics



# Citace

Zdeněk Jurka: Návrhář a generátor budov, bakalářská práce, Brno, FIT VUT v Brně, 2010

# Návrhář a generátor budov

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Přibyla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdeněk Jurka
19. května 2010

## Poděkování

Rád bych poděkoval panu Ing. Jaroslavu Přibylovi za pomoc a odborné vedení při tvorbě této práce.

# Obsah

# Kapitola 1

# Úvod

Pokud tvoříme plány budov, můžeme mít na mysli plánky pro orientaci budovou, plánky, které nám pomohou při rozmisťování objektů v místnostech nebo kompletní stavební plány. V případě, že tyto plány budeme vytvářet na počítači, použijeme podle našich požadavků příslušný software, který nám umožní snadno dosáhnout námi požadovaných výsledků.

Tato práce se zabývá návrhem a implementací jednoduchého editoru pro vytváření orientačních plánků budov. V kapitole dvě se budeme zabývat analýzou problému, ukázkou již hotových řešení a zařazení našeho programu v rámci těchto řešení. Ve třetí kapitole je rozebrána základní teorie z oblasti počítačové grafiky, která byla využita při tvorbě tohoto projektu. Jedná se o rozdíly mezi rastrovou a vektorovou grafikou, 2D transformace a práci s vektory. Čtvrtá kapitola popisuje prostředky zvolené pro tvorbu aplikace. V kapitole pět je rozebrán návrh celé aplikace. Jsou zde definovány cílové funkce programu. Dále se zde nachází popis reprezentace dat, objektový návrh aplikace a návrh grafického uživatelského rozhraní. Kapitola šest prochází jednotlivé kroky a principy použité při implementaci programu. Popisují se zde postupně základní algoritmy použité v aplikaci jako je přichytávání objektů, vykreslování a reakce na vstup uživatele. Sedmá kapitola popisuje možné rozšíření aplikace o generování 3D modelu a nastiňuje možný postup řešení.

# Kapitola 2

# Analýza

Cílem projektu je vytvořit editor pro tvorbu 2D plánků budov s možností v budoucnosti rozšířit aplikaci o generování 3D modelu objektu. Naše aplikace by měla umožnit vytvoření plánku jednotlivých pater budovy a jejich provázání. Editor by měl umožnit práci podle zvoleného vzoru v podobě obrázku. Účelem aplikace je jednoduché a rychlé vytvoření orientačního plánku celé budovy. Orientačním plánkem je myšlen plánek bez vyznačených rozměrů, který nemusí být zcela shodný s předlohou.

## 2.1 Existující řešení

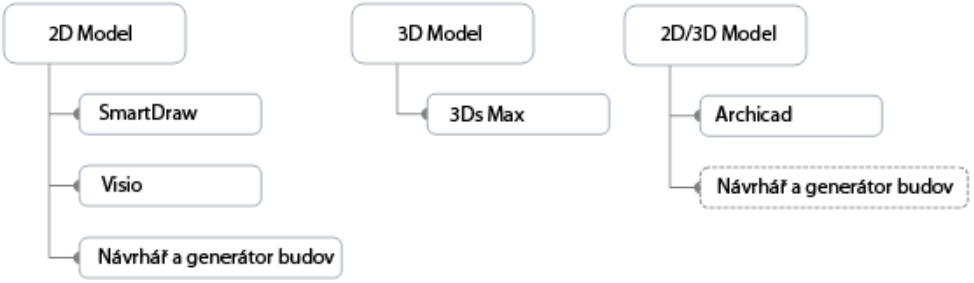
Aplikací, které řeší stejný problém, existuje celá řada. Od jednoduchých, které umožňují nakreslení jednoduchého 2D plánku, po aplikace, které umožňují navrhnout celý objekt včetně konstrukčních podrobností a vygenerování 3D modelu. Jednoduché vytvoření plánku umožňuje například SmartDraw [6] nebo Microsoft Office Visio [4]. Mezi aplikace pro pokročilejší návrh budov patří například Archicad [2]. Pro vizuální návrh 3D modelu budov je možné použít 3D modelovací programy jako je například Autodesk 3Ds Max [1]. Ve všech zde uvedených případech se jedná o komerční aplikace.

U jednodušších aplikací jako je SmartDraw a Visio se jedná o multifunkční aplikace, které umožňují vytvářet velké množství různých diagramů, grafů a plánků z různých oblastí. V případě tvorby plánku budov umožňují vytvořit jednoduchý plánek podlaží budovy a umístit do plánku další objekty jako jsou stoly, skříně apod. V tomto případě se jedná spíše o plánky orientační s možností rozvržení jednotlivých objektů v místnosti.

Pokročilé aplikace jako je Archicad dovolují pokročilý návrh celé budovy včetně možnosti definování materiálů jednotlivých částí budovy apod. Zde se již jedná o kompletní návrh budovy včetně všech technických podrobností. Dále dovolují zobrazení 3D modelu celého objektu včetně volby vizuálního vzhledu jednotlivých částí budovy. Tyto aplikace jsou určeny pro profesionální použití v oblasti architektury a umožňují vytvářet kompletní stavební plány.

V případě pouhého vizuálního návrhu bez vazby na tvorbu plánů je možné využít 3D modelovací software jako již zmíněné 3Ds Max. Tento druh softwaru umožňuje pokročilý vizuální návrh objektu s velice realistickým výstupem v podobě 3D scény.

Mezi těmito aplikacemi se naše aplikace řadí mezi jednoduché editory pro kreslení orientačních plánků budov viz. obrázek 2.1.




Obrázek 2.1: Rozdělení editorů podle práce s 2D nebo 3D modelem.

## 2.2 Postup při řešení

Jako první část při řešení by bylo vhodné stanovit si funkčnost a cíle, které musí editor splňovat. Na základě takto stanovených cílů je možné vybrat vhodné implementační prostředí, které umožní jejich snadnou implementaci. Před samotnou implementací je však nutné provést návrh aplikace. V rámci návrhu aplikace je zapotřebí vytvořit objektový model aplikace, určit formát dat, se kterými se bude pracovat, a navrhnout uživatelské rozhraní. Při návrhu objektového modelu je dobré se zaměřit na jeho snadnou použitelnost a údržbu takového modelu při pozdějších rozšířeních. Po návrhu a výběru implementačního prostředí je možné přistoupit k samotné implementaci projektu.

# Kapitola 3

# Teorie

Tato kapitola se zabývá teorií z oblasti počítačové grafiky pro objasnění principů použitých při tvorbě aplikace. Tato kapitola převážně čerpá z [13].

## 3.1 Vektorová a rastrová grafika

Počítačovou grafiku můžeme z pohledu reprezentace grafických dat rozdělit na rastrovou a vektorovou.

### 3.1.1 Rastrová grafika

V případě rastrové (bitmapové) grafiky je obraz reprezentován maticí bodů (2D/3D), z nichž každý nese informaci o barvě bodu. Tuto reprezentaci můžeme získat manuálně (pomocí bitmapového grafického editoru), syntézou, snímáním (scanner, digitální fotoaparát, ...). [10]

Informace o barvě může mít různou podobu:

- Každý bod je reprezentován jako jeden bit, jedná se o monochromatický obraz, kde hodnota bitu určuje jednu ze dvou barev např.: černá nebo bílá.
- Každý bod nese ukazatel do palety (mapy) barev, jedná se o indexový mód. Pomocí ukazatele se z palety vybere barva konkrétního bodu.
- Obraz je reprezentován ve stupních šedi. Každý bod může přímo nést informaci o odstínu šedi nebo může ukazovat do palety.
- Každý pixel nese informaci o třech barvách, nejčastěji v modelu RGB. Může obsahovat přímo barevné hodnoty – True color, nebo indexy do jednotlivých barevných palet (pro každý kanál jedna) – direct color.

V rastrové grafice se uchovává pouze informace o podobě objektu (tvar, barva, ...), neuchovávají se však informace o objektu samotném. Tento princip znemožňuje jednoduchou úpravu jednotlivých objektů. Obraz je možno upravovat pouze po jednotlivých bodech. [10]

Vzhledem k tomu, že většina zobrazovacích zařízení funguje na rastrovém principu, je zapotřebí jiné grafické reprezentace převést na rastrovou. Tento proces se nazývá rasterizace.

### 3.1.2 Vektorová grafika

Při vektorové reprezentaci se grafické informace uchovávají ve formě vektorových entit (úsečky, kružnice, křivky, ...). Uchovávají se zde informace o popisu zobrazených objektů. Jednotlivé entity jsou popsány pomocí vlastností (barva, styl, ...) a parametrů (poloha, velikost, orientace, ...). Tvar jednotlivých entit je popsán geometricky. Úsečka je definována počátečním bodem, koncovým bodem a přímkou, kružnice souřadnicemi středu a poloměrem, atd.

Díky tomu, že se uchovávají informace o objektech, umožňuje tato reprezentace na rozdíl od rastrové jednoduchou úpravu objektů jako je změna barvy a tloušťky čáry, pozice v obrazu, tvaru, atd.

Pokud se objekty zobrazují na rastrovém zobrazovacím zařízení (většina v současnosti používaných), musí se před každým zobrazením převést na svou rastrovou reprezentaci – rasterizace. [10]

## 3.2 2D Transformace

Transformace můžeme rozdělit na lineární a nelineární. U lineárních transformací zůstává koeficient transformace (vektor posunutí, koeficient zvětšení, ...) stejný pro celý objekt na který se transformace aplikuje. U nelineárních transformací se tento koeficient může měnit například v závislosti na poloze bodu v souřadném systému. [11]

Mezi lineární transformace patří všechny základní transformace jako je posunutí, otočení, změna měřítka a zkosení, dále pak transformace vzniklé jejich složením. S nelineárními transformacemi se můžeme setkat například u složitých změn tvaru objektů a různých grafických efektů.

Pro zjednodušení výpočtů transformací se využívá reprezentace bodů pomocí homogenních souřadnic. Výhodou této reprezentace je, že umožňuje vyjádření nejčastějších lineárních transformací pomocí jediné transformační matice. Skládání transformací je v tomto případě možno realizovat pomocí násobení jednotlivých transformačních matic a inverzní transformaci lze vyjádřit jako inverzní matici.

Pro bod $P$ určený kartézskými souřadnicemi $[X, Y]$ představuje uspořádaná trojice $[x, y, w]$ homogenní souřadnice, pokud platí:

$$X = \frac{x}{w}, \qquad Y = \frac{y}{w}, \qquad w \neq 0$$

### 3.2.1 Posunutí

Transformace posunutí (translace) bodu P je dána vektorem posunutí $\vec{p} = (X_t, Y_t)$. Matice transformace posunutí $\mathbf{T}$ a matice inverzní transformace $\mathbf{T^{-1}}$ mají tvar:

$$\mathbf{T}(X_t, Y_t) = \begin{bmatrix} 1 & 0 & X_t \\ 0 & 1 & Y_t \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{T^{-1}}(X_t, Y_t) = \begin{bmatrix} 1 & 0 & -X_t \\ 0 & 1 & -Y_t \\ 0 & 0 & 1 \end{bmatrix}$$

### 3.2.2 Otáčení

Otočením bodu $P$ kolem počátku souřadného systému $O = [0, 0]$ o úhel $\alpha$ získáme posunutý bod $P'$ se souřadnicemi:

$$X' = X \cos\alpha - Y \sin\alpha$$

$$Y' = X \sin\alpha + Y \cos\alpha$$

Matice této transformace $\mathbf{R}$ a matice inverzní transformace $\mathbf{R^{-1}}$ mají tvar:

$$\mathbf{R}(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{R^{-1}}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### 3.2.3 Změna měřítka

Tato transformace má vliv na polohu i velikost transformovaného objektu ve směru souřadnicových os. Výsledná podoba transformace je dána hodnotou koeficientu změny měřítka $s$. Pokud je jeho absolutní hodnota v intervalu $(0, 1)$, dochází ke zmenšení a přiblížení objektu k počátku souřadného systému. V případě, že je absolutní hodnota koeficientu větší než jedna, dojde ke zvětšení objektu. Záporné znaménko má za následek zmenšení nebo zvětšení v opačném směru.

Matice změny měřítka $\mathbf{S}$ a matice inverzní transformace $\mathbf{S^{-1}}$, kde $s_x$ označuje koeficient změny měřítka ve směru osy $x$ a $s_y$ ve směru osy $y$, mají tvar:

$$\mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{S^{-1}}(s_x, s_y) = \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### 3.2.4 Zkosení

Transformace zkosení ve směru osy $x$ je dána koeficientem zkosení $sh_x$ a ve směru osy $y$ koeficientem $sh_y$. Transformační matice mají tvar:

$$\mathbf{Sh_x}(sh_x) = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{Sh_x^{-1}}(sh_x) = \begin{bmatrix} 1 & -sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{Sh_y}(sh_y) = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{Sh_y^{-1}}(sh_y) = \begin{bmatrix} 1 & 0 & 0 \\ -sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## 3.3 Vektory

Vektor je veličina, která má kromě své velikosti také směr. Vektor $\vec{u} = (u_1, u_2)$ v dvourozměrném prostoru z bodu $A = [a_x, a_y]$ do bodu $B = [b_x, b_y]$ je určen jako:

$$\vec{u} = (b_x - a_x, b_y - a_y)$$

Velikost vektoru je dána:

$$|\vec{u}| = \sqrt{u_1^2 + u_2^2}$$

Sčítání vektorů:

$$\vec{w} = \vec{u} + \vec{v} = (u_1 + v_1, u_2 + v_2)$$

Odečítání vektorů:

$$\vec{w} = \vec{u} - \vec{v} = (u_1 - v_1, u_2 - v_2)$$

Opačný vektor:

$$-\vec{u} = (-u_1, -u_2)$$

Násobení vektoru skalárem:

$$u_1 = u_1 \cdot A$$

$$u_2 = u_2 \cdot A$$

Skalární součin:

$$\vec{u} \cdot \vec{v} = u_1 \cdot v_1 + u_2 \cdot v_2$$

Jednotkový vektor:

$$u_1 = \frac{u_1}{|\vec{u}|}$$

$$u_2 = \frac{u_2}{|\vec{u}|}$$

# Kapitola 4

# Použité prostředky

Tato kapitola čerpá z [12].

## 4.1 Implementace

Program je implementován v jazyce C# s využitím .NET Frameworku. Jazyk C# je objektově orientovaný a typově bezpečný programovací jazyk, který je odvozen od jazyků C, C++ a Java. Tento jazyk je speciálně vytvořen pro technologii .NET, jedná se však o samostatný jazyk, který není sám o sobě součástí platformy .NET.

Základem platformy .NET je její běhové prostředí označované jako modul CLR (Common Language Runtime). Kód spuštěný pod kontrolou tohoto modulu bývá označován jako řízený kód (managed code). Libovolný kód spouštěný v tomto běhovém prostředí podléhá dvoufázovému procesu překladu. Nejdříve se zdrojový kód přeloží do jazyka IL (nízkoúrovňový jazyk s jednoduchou syntaxí). Překlad do nativního kódu pro cílovou platformu probíhá až v okamžiku volání dané části kódu. Tento způsob překladu se označuje jako JIT (just in time compilation). Hlavní výhodou kódu spuštěného v rámci modulu CLR je možnost využití knihovny základních tříd platformy .NET.

Knihovna tříd .NET Frameworku obsahuje třídy grafického uživatelského rozhraní systému Windows, třídy pro grafický výstup, třídy pro práci se sítí a souborovým systémem, třídy pro přístup k databázím a pro možnost dotazování nad daty a mnoho dalších. Díky této knihovně tříd bylo nutné při vývoji aplikace navrhnout pouze minimum vlastních tříd. Kvůli těmto vlastnostem jazyka C# a knihovny tříd .NET Frameworku jsem zvolil toto implementační prostředí.

## 4.2 Vývojové prostředí

Editor byl vyvíjen v prostředí Microsoft Visual Studio 2008. Jedná se o integrované vývojové prostředí (IDE), které obsahuje rozsáhlou podporu pro tvorbu aplikací v jazyce C# pod platformou .NET. Toto prostředí umožňuje rozsáhlou správu projektů, má vestavěné pokročilé ladicí prostředí a poskytuje automatické doplňování a generování kódu. Dále zahrnuje prostředí pro návrh a tvorbu uživatelských rozhraní.

# Kapitola 5

# Návrh aplikace

Celá aplikace je založená na objektovém modelu a je vytvářena jako okenní aplikace pro systémy Windows. Při vytváření objektového modelu aplikace byla kladena snaha na maximální využití prostředků zvoleného implementačního prostředí.

## 5.1 Cíle

Aplikace by měla umožnit snadnou a co nejrychlejší tvorbu 2D plánků budov. Editor musí dovolit rozdělit budovu do pater a vytvořit plánek pro každé patro zvlášť a propojení pater pomocí schodišť. Dále by aplikace měla poskytnout prostředky pro jednoduchou editaci již vytvořených objektů jako je posun v rámci plánku, změna velikosti, případně odstranění objektu. Pro tyto úpravy by bylo vhodné implementovat funkci pro vrácení těchto akcí zpět. Dále by aplikace měla zajistit základní test kolizí mezi vytvářenými objekty pro zjednodušení vkládání objektů do plánku. Editor by měl umožnit zobrazit na kreslicím plátně mřížku pro zjednodušení orientace při kreslení, případně předlohu v podobě již hotového obrázku. Aplikace musí dovolit jak uložení vytvořeného plánku do souboru, ze kterého bude schopna všechny již vytvořené objekty znovu načíst, zobrazit a editovat, tak export do různých grafických formátů (jpeg, bmp, apod.). Editor by měl být navržen tak, aby v budoucnosti umožnil přidání exportu do 3D modelu budovy. Na základě těchto požadavků byl vytvořen objektový model celé aplikace a navrženo grafické uživatelské rozhraní.

## 5.2 Reprezentace dat

### 5.2.1 Grafická data

Grafická data u grafických objektů jako souřadnice počátku a konce objektu, velikost, natočení a tvar objektu jsou udržována ve vektorové podobě. Díky této reprezentaci grafických dat je v aplikaci možné snadno realizovat operace pro práci s objekty jako jejich posun, změnu velikosti, smazání objektu, změnu měřítka apod. Před každým vykreslením je však potřeba provést rasterizaci těchto objektů.

### 5.2.2 Formát ukládaných dat

Stav aplikace a grafická data jsou ukládána do formátu XML. Tento formát díky velké podpoře v .NET frameworku umožňuje snadné uložení a znovunačtení dat, případně mož-

nou editaci těchto souborů mimo tuto aplikaci. Dále umožňuje v případě potřeby snadnou přenositelnost těchto dat mezi různými aplikacemi. [9]

Aplikace umožňuje export jednotlivých částí budovy do rastrového obrázku. Pro tyto účely byl vybrán formát bmp, jpeg a png. Jeden z nich si může uživatel zvolit ve chvíli exportu dané části plánku do rastrového obrázku.

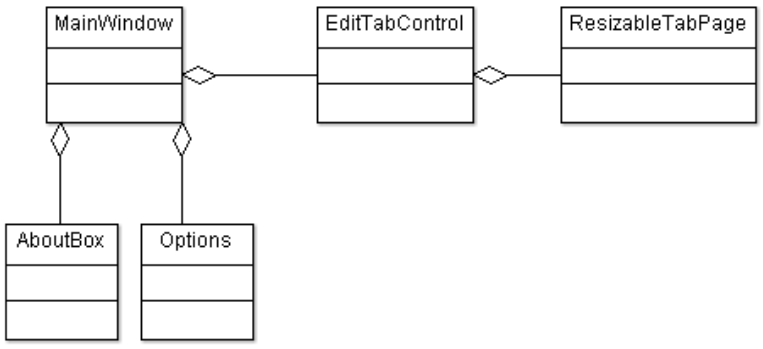
## 5.3 Objektový model

Cílem při vytváření objektového návrhu aplikace bylo vytvoření jednoduchého a snadno rozšiřitelného modelu. Celý model je rozdělen do čtyř logických celků, které reprezentují jednotlivé části aplikace. V této podkapitole jsou ukázány jednotlivé části diagramu tříd. Pro zachování přehlednosti a znázornění hierarchie a návaznosti tříd jsou z nich vynechány popisy jednotlivých metod a vlastností.

### 5.3.1 Uživatelské rozhraní

Základem této části je třída hlavního okna aplikace *MainWindow*, která se stará o komunikaci s uživatelem. Dále jsou zde definovány třídy *AboutBox* a *Options*, které třída MainWindow využívá. Tyto třídy dědí od třídy *Form* definované v .NET Frameworku. Návaznost jednotlivých tříd je patrná ze zjednodušeného diagramu tříd uživatelského rozhraní viz. obrázek 5.1. Grafický vzhled některých tříd je ukázán na obrázku 5.2.

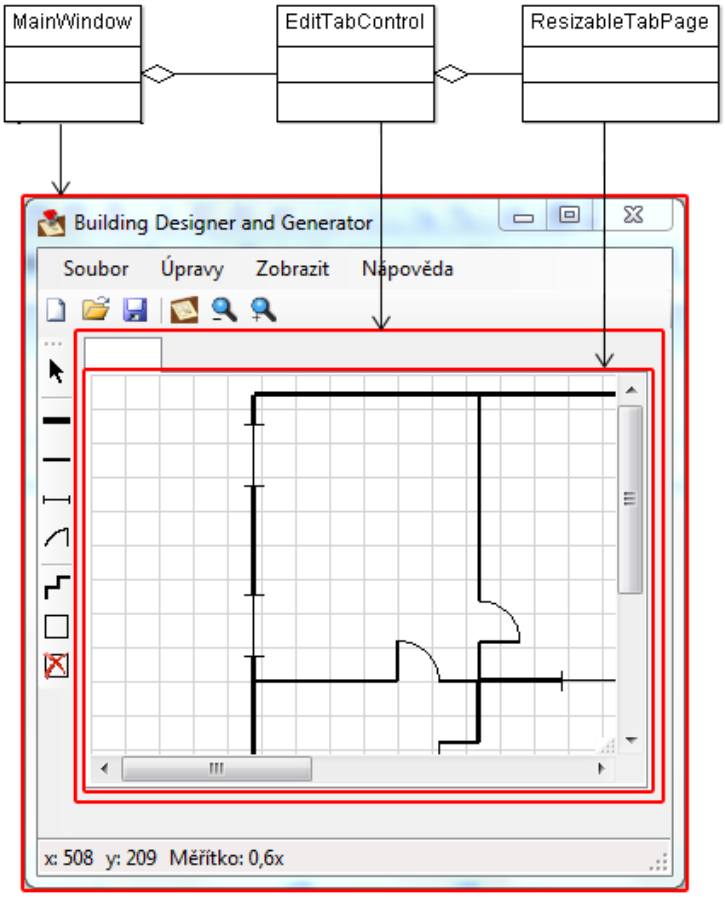
Třída MainWindow dále používá třídy definované v .NET Frameworku. Jedná se o třídy reprezentující grafické prvky uživatelského rozhraní jako jsou dialogy pro otevření a uložení souboru, dále hlavní menu aplikace a panely nástrojů. Mezi další zde použité třídy patří *ResizableTabPage* a *EditTabControl*, které rozšiřují a upravují komponenty uživatelského rozhraní. Tyto dvě třídy jsou vloženy do samostatné knihovny ovládacích prvků.




Obrázek 5.1: Diagram tříd uživatelského rozhraní.

### 5.3.2 Logika aplikace

Celou aplikaci řídí statická třída *Editor*. Tato třída přebírá informace od uživatele, které jí předávají třídy uživatelského rozhraní. Na základě těchto informací se řídí vykreslování. Jsou zde definovány metody pro obsluhu kurzoru myši při kreslení objektů. MainWindow zde na základě podnětů uživatele nastavuje parametry editoru jako je druh vybraného nástroje pro kreslení, velikost a zobrazení mřížky, nastavení obrázku jako předlohy a další.

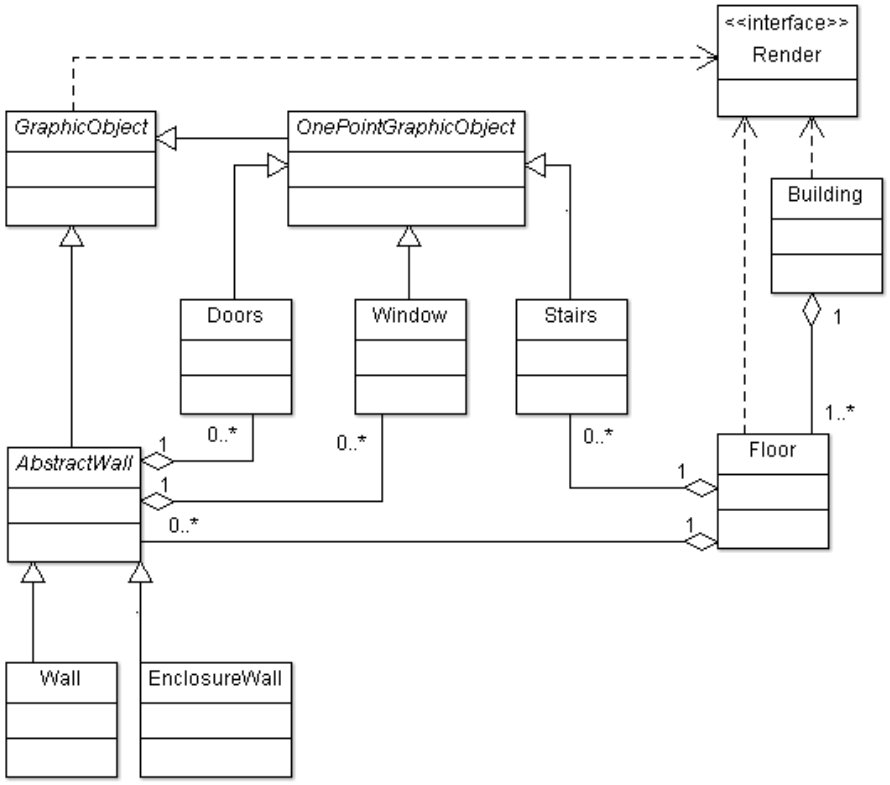



Obrázek 5.2: Grafická reprezentace některých tříd GUI.

### 5.3.3 Grafické objekty

Pro všechny grafické objekty je vytvořeno rozhraní *Render* které definuje metodu paint2D pro vykreslení prvku. Třídy zde reprezentují jednotlivé části budovy. Třídy Building a Floor reprezentují logickou strukturu budovy, třídy Window, Doors, Wall, EnclosureWall a Stairs fyzické prvky budovy. Dále jsou zde abstraktní třídy GraphicObject, OnePointGraphicObject a abstractWall, které spojují společné části jednotlivých objektů. Všechny třídy definují metodu pro převod na XML reprezentaci objektu. Vztahy mezi třídami jsou patrné z diagramu tříd grafických prvků viz obrázek 5.3.

- **Abstraktní třída GraphicObject** je společná pro všechny objekty tvořící fyzickou část budovy a definuje vlastnosti jako počáteční a koncové souřadnice objektu, jeho velikost a orientaci v prostoru. Předepisuje také metody pro pohyb a editaci objektu. Mezi tyto objekty patří zdi, obvodové zdi, okna, dveře a schody. Dále jsou zde implementovány metody pro řazení objektů a natáčení objektů.
- **Abstraktní třída OnePointGraphicObject** rozšiřuje GraphicObject pro objekty, které jsou definovány pouze počátečním bodem a úhlem natočení. Mezi tyto objekty patří dveře, okno a schody. Jsou zde implementovány metody pro pohyb objektem a změnu velikosti objektu. Dále je zde překryta vlastnost koncového bodu, který není zadán přímo, ale je dopočítán.




Obrázek 5.3: Diagram tříd grafických objektů.

- **Abstraktní třída abstractWall** rozšiřuje GraphicObject pro zeď a obvodovou zeď. Nachází se zde metody pro pohyb objektu a editaci tvaru objektu. Je zde udržován seznam objektů ve zdi. Tento seznam může obsahovat okna a dveře. Jsou zde metody pro přidání těchto objektů do zdi a metoda pro kontrolu platnosti umístění prvku pro zeď.
- **Třídy Window, Doors, Stairs, Wall a EnclosureWalls** implementují metodu paint, která vykreslí grafickou reprezentaci objektu.
- **Třída Building** představuje celou budovu a udržuje seznam všech vytvořených pater a metody pro práci s nimi.
- **Třída Floor** představuje jedno podlaží budovy. Nachází se zde seznam všech zdí v podlaží (obvodových i vnitřních), dále seznamy schodišť (schodiště do vyššího patra a schodiště do nižšího patra). Nachází se zde metody pro přidání a kontrolu platnosti umístění těchto objektů.

### 5.3.4 Podpůrné třídy

Tyto třídy byly navrženy pro zjednodušení implementace celého editoru.

- **Třída Vector** reprezentuje vektor v dvourozměrném prostoru. Jsou zde definovány metody pro práci s vektory jako je jejich násobení a sčítání. Dále metoda pro vytvoření normovaného vektoru a normálového vektoru.

- **Třída Scale** představuje měřítko editoru a zapouzdřuje práci s ním.
- **Třída Action** představuje akci pro umožnění kroku zpět v editoru. Je v ní uložen stav objektu před jeho změnou a akce, která se s objektem provedla.
- **Třída Fce** obsahuje pomocné metody, které jsou využívány v různých částech aplikace. Jedná se například o metody pro zjištění polohy bodu a úsečky, přichycení bodu k přímce nebo zjištění kolize bodu a obecného obdélníka.
- **Výčtový typ eTools** obsahuje výčet všech nástrojů editoru (zeď, okno, schody, ...), **eClick** pozici kliknutí na objekt (počátek, konec), **eAction** akce pro krok zpět.

## 5.4 Grafické uživatelské rozhraní

Uživatelské rozhraní bylo navrhováno s ohledem na jednoduchost a rychlost používání celé aplikace. Aplikace má hlavní okno, ze kterého se v případě potřeby zobrazují modální dialogová okna. [14] V souvislosti s uživatelským rozhraním se velmi často zmiňuje pojem použitelnost (usability). Následující definice je převzatá z [7].

**Použitelnost** (usability) je vlastnost produktu, která udává jak jednoduše se může uživatel naučit pracovat s produktem a dále používat tento produkt pro splnění svých cílů a jeho spokojenost s tímto procesem. Použitelnost měří úroveň práce s produktem. Použitelnost se skládá z několika částí:

- **Pochopitelnost** – jak rychle se uživatel, který uživatelské rozhraní nikdy neviděl, může naučit základní úkony?
- **Efektivita použití** – jak rychle dokáže uživatel, který se naučil ovládat uživatelské rozhraní, splnit zadané úkoly?
- **Zapamatovatelnost** – pokud uživatel systém delší dobu nepoužívá, dokáže si zapamatovat tolik, aby příště mohl systém znovu efektivně používat?
- **Počet a závažnost chyb** – jak často uživatel udělá chybu, jak závažná tato chyba je a jak se dokáže s chybou vypořádat?
- **Subjektivní dojem** – jak je uživatel spokojen s používáním a ovládáním systému?

S ohledem na tato fakta byl při návrhu a tvorbě uživatelského rozhraní kladen důraz na jeho jednoduchost, intuitivnost a rychlost použití. Testování uživatelského rozhraní bylo zjednodušeno na testování na několika uživatelích. Na základě jejich zkušeností byly do programu přidány další funkce pro zjednodušení a zrychlení práce. Seriózní vyhodnocení použitelnosti navržené aplikace je samostatným oborem přesahujícím rámec této práce.

# Kapitola 6

# Implementace

Při implementaci bylo čerpáno z *Microsoft Developer Network*, kde se nachází popis jednotlivách tříd v .NET Frameworku [5].

## 6.1 Podpůrné třídy

Mezi podpůrné třídy patří třída Fce, Vector, Action a Scale. Dále sem můžeme zařadit definované výčtové typy eTool, eClick a eAction. Metody a účel třídy Fce je kvůli její složitosti a důležitosti vysvětlen v samostatné části.

### 6.1.1 Práce s vektory

Třída *Vector* zapouzdřuje práci s vektory pro potřeby aplikace. Implementované metody pro skalární součin dvou vektorů, vynásobení vektoru skalárem a přičtení nebo odečtení vektoru k bodu jsou implementované jako přetížení daných operátorů. Přetížení operátorů usnadňuje zápis kódu při práci s vektory. Tato třída má veřejné vlastnosti určené pouze ke čtení, které dovolují přístup k jednotlivým složkám vektoru, bodům mezi kterými je vytvořen a délce vektoru. Dále obsahuje obyčejné metody, které daný vektor převedou na normalizovaný, opačný nebo normálový. Vzorce použité pro práci s vektory jsou uvedeny v části 3.3.

### 6.1.2 Historie akcí

Třída *Action* představuje akci pro krok zpět v editoru. Pro vrácení změn na vytvořeném objektu jako je pohyb nebo změna velikosti si tato třída udržuje informace o grafickém objektu před jeho změnou. V případě vyvolání akce zpět se nastaví vlastnosti objektu zpět na původní. Pokud se akce zpět vytvoří při vytvoření objektu, zavolá se při vyvolání pouze metoda Remove. Pokud je akce vytvořena při odstranění objektu, zavolá se při vyvolání akce metoda add nad příslušným grafickým objektem. Konstruktoru třídy se předává jako parametr grafický objekt, na který se akce váže, a druh akce. Druhý konstruktor přebírá jako první parametr seznam grafických objektů, druhým parametrem je akce. Toho se využívá při výběru a úpravě více objektů. Pro každý grafický objekt se zde vytvoří jedna třída Action a uloží se v seznamu v původní instanci třídy Action. Ta pak při vyvolání akce prochází tento seznam a volá nad jednotlivými objekty akci pro krok zpět. Seznam těchto akcí je udržován v zásobníku ve třídě Editor. Obdobně je realizovaná funkce vpřed, která se vytváří při zavolání kroku zpět.

### 6.1.3 Změna měřítka

Pro měřítko editoru je vytvořena třída *Scale*. Tato třída umožňuje pomocí metod *ZoomIn* a *ZoomOut* změnit nastavení měřítka použitého pro vykreslení objektů. Pro zjednodušení práce s touto třídou jsou přetíženy operátory pro násobení a dělení. Operátor násobení je přetížen pro jednoduché vynásobení souřadnic objektů při vykreslování. Operátor pro dělení se používá pro získání odpovídajících souřadnic polohy myši.

### 6.1.4 Výčtové typy

Výčtový typ *eTool* obsahuje výčet všech možných nástrojů editoru jako je nástroj pro vložení okna, dveří, zdí nebo nástroj pro pohyb nebo změnu velikosti objektu. Podle těchto nástrojů se řídí akce vyvolaná například při kliknutí nebo pohybu myší. Údaj o aktuálně vybraném nástroji je ve vlastnosti tool třídy Editor.

Výčtový typ *eAction* obsahuje seznam všech podporovaných akcí pro historii akcí. Na základě druhu akce se v třídě Action při vyvolání události zpět řídí činnost této metody.

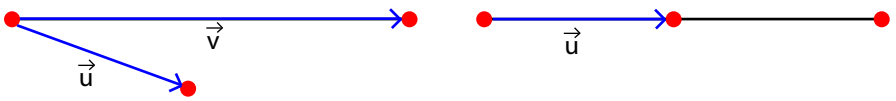
Výčtový typ *eClick* se používá pro předání informace o tom, která část grafického objektu se upravuje, zda jeho počátek nebo konec.

## 6.2 Podpůrné funkce

Pomocné funkce jsou definovány jako statické metody ve třídě *Fce*. Tyto metody jsou používány v různých částech aplikace a proto byly umístěny do samostatné třídy jako statické metody.

### 6.2.1 Zjištění polohy bodu a úsečky

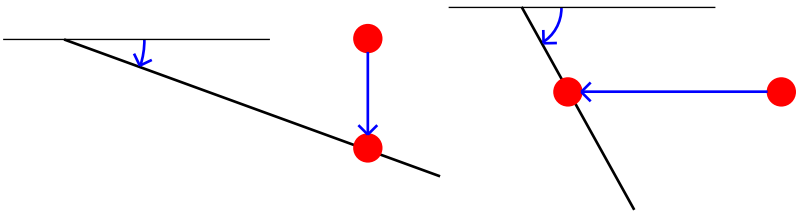
Pro zjištění, jestli bod leží na úsečce, slouží metoda *isOnLine*. Tato metoda přijímá jako parametry krajní body úsečky a bod, pro který se testuje poloha vůči přímce. Nejdříve se určí vektor z počátečního do koncového bodu úsečky a z počátečního bodu do testovaného bodu. Pak se tyto vektory normalizují a určí se jejich skalární součin. Pokud je skalární součin roven jedné, jsou vektory rovnoběžné a leží na stejné přímce viz. obrázek 6.1. Dále je nutné otestovat, jestli bod leží mezi krajními body úsečky.




Obrázek 6.1: Zjištění polohy bodu a úsečky.

### 6.2.2 Přichycení bodu k přímce

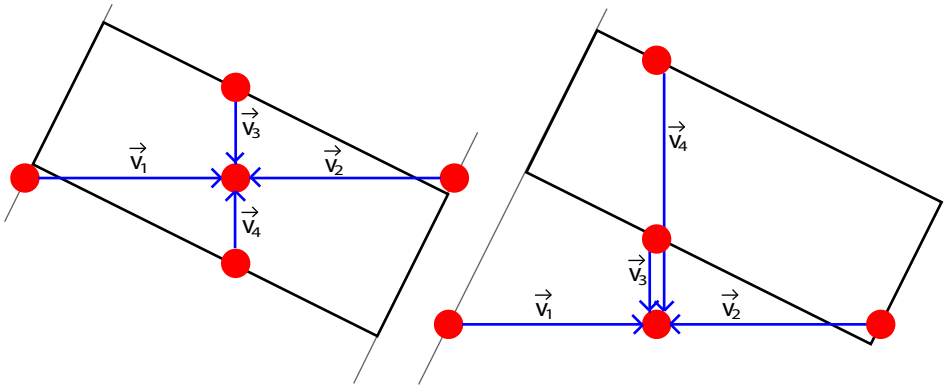
Pro přichycení bodu k přímce je implementována metoda *snapToLine*, které se jako parametry předávají dva body určující přímku, a bod, který se má k přímce přichytit. V metodě se určí jednotlivé složky ve směrnicovém tvaru přímky. V závislosti na natočení přímky se do rovnice dosadí jedna souřadnice přichytávaného bodu a druhá se dopočítá, čímž se bod umístí na přímku. Pokud je úhel natočení přímky menší než 45°, dopočítá se x-ová souřadnice, pokud je větší, dopočítá se y-ová souřadnice viz. obrázek 6.2. Tím se zajistí nejpřesnější přichycení bodu.



Obrázek 6.2: Přichycení bodu k přímce.

### 6.2.3 Zjištění polohy bodu a obdélníku

Pro zjištění, jestli bod leží uvnitř obdélníku, slouží metoda *Colision*. Tato metoda přebírá jako parametry čtyři body, které reprezentují rohy obdélníku, a bod, pro který se testuje kolize s obdélníkem. Nejdříve se tento bod přichytí k jednotlivým přímkám, na kterých leží úsečky tvořící obdélník. Poté se určí vektory z těchto bodů do zkoumaného bodu. Pokud jsou znaménka protilehlých vektorů různá, leží bod uvnitř obdélníku, pokud jsou stejná, leží bod vně jak je znázorněno na obrázku 6.3.




Obrázek 6.3: Test polohy bodu a obdélníku.

## 6.3 Vykreslování

Vykreslování používá třídy GDI+. Základem je třída *Graphics*, která zapouzdřuje práci s kontextem zařízení, do kterého se vykresluje. Instanci této třídy je možné získat zavoláním metody *CreateGraphics* u prvku grafického rozhraní odvozeného od třídy Control. Další možností je statická metoda *FromImage* třídy Graphics, která vytvoří instanci této třídy pro kreslení do obrázku reprezentovaného třídou Image. Tato třída obsahuje metody pro základní geometrické transformace (posunutí, otočení a změna měřítka) a pro vykreslení základních geometrických primitiv jako jsou čáry, obdélníky, kružnice, elipsy a křivky.

### 6.3.1 Princip

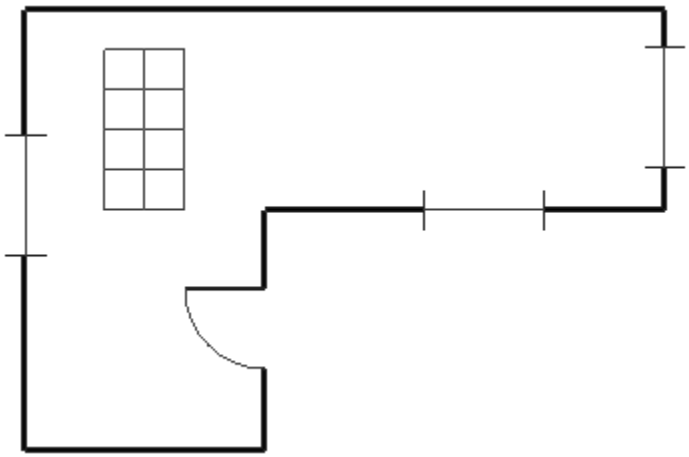
Vykreslování probíhá do prvku ResizableTabPage, u kterého je pomocí metody *SetStyle* povolena vlastnost *ControlStyles.OptimizedDoubleBuffer*, která u daného prvku zapíná double buffering pro plynulé vykreslování. Pro plynulé vykreslování animací je potřeba zaručit

překreslování prvku v pevně daných intervalech. Žádost o překreslení je generována pomocí čítače každých 20 ms. Překreslení se děje na základě zavolání metody *Invalidate*.

Vykreslování probíhá v metodě *Paint*, která je definovaná ve třídě Editor. Tato metoda se volá na základě události Paint třídy ResizableTabPage, která je generována ve chvíli, kdy přijde žádost na překreslení prvku. Nejdříve se vykreslí mřížka, pokud je tato možnost v editoru povolena. Následuje vykreslení grafické reprezentace vybraného nástroje. Pokud se jedná o zeď nebo obvodovou zeď, vykreslí se při stisknutém tlačítku myší čára, v případě okna, dveří nebo schodiště se vykreslí grafická reprezentace tohoto objektu na pozici kurzoru myši. V posledním kroku se vykreslí aktivní patro budovy zavoláním metody Paint nad třídou představující tvořenou budovu. Tato metoda zavolá stejnou metodu nad aktivním patrem budovy. Zde se tato metoda zavolá u všech objektů, které dané patro obsahuje. Tyto metody si navzájem jako parametr předávají instanci třídy Graphics, ve které probíhá vykreslování.

### 6.3.2 Vykreslování grafických objektů

U objektů, které reprezentují okno, dveře nebo schodiště, se nejdříve zavolá metoda pro transformaci posunutí do počátečního bodu a následně transformace rotace pro natočení do požadovaného úhlu. Následuje vlastní vykreslení grafické reprezentace objektu. V případě zdi nebo obvodové zdi se nejdříve vykreslí všechny prvky, které zeď obsahuje (okna, dveře), a následně se vykreslí úsečky mezi těmito objekty a začátkem a koncem zdi. Pro vykreslení prvku se používají metody DrawLine, která vykreslí přímku určenou dvěma body a v případě dveří ještě DrawArc, která vykreslí část elipsy určenou obdélníkem a počátečním a koncovým úhlem. Vzhled těchto objektů je ukázán na obrázku 6.4.



Obrázek 6.4: Grafické objekty.

### 6.3.3 Změna měřítka

Před samotným vykreslením se jednotlivé souřadnice bodů předávané do metod pro vykreslení jednotlivých geometrických primitiv vynásobí třídou Scale,která reprezentuje měřítko editoru. Tato třída se nachází ve třídě Editor jako její vlastnost. Tímto je možné přiblížit nebo oddálit vykreslované objekty bez přímé změny jejich souřadnic.

## 6.4 Vytváření objektů

Pro vytváření a upravování objektů jsou ve třídě Editor definovány metody, které reagují na události myši a klávesnice. Metody reagují na pohyb a stisknutí tlačítka myši nad kreslicí oblastí editoru, dále pak na stisknutí definovaných kláves. Při pohybu myši jsou ukládány údaje o poslední poloze myši a o pozici myši při stisknutí tlačítka.

Uživatel nejdříve vybere druh objektu, který chce vložit v panelu nástrojů. Na základě této akce třída MainWindow nastaví v editoru vlastnost tool na požadovaný druh objektu.

### 6.4.1 Zdi a obvodové zdi

Při vkládání obvodové nebo vnitřní zdi se při stisknutí levého tlačítka myši uloží pozice kliknutí myši a v případě zapnutého přichytávání (k objektu nebo k mřížce) se pozice tohoto bodu patřičně upraví. Zároveň se v editoru nastaví vlastnost *wallIn* na true. Na základě této vlastnosti se vykreslí čára od souřadnice kliknutí k současné poloze kurzoru, která reprezentuje vkládanou zeď. Při dalším stisknutí tlačítka se zavolá metoda *addEnclosureWall* pro vložení obvodové zdi, nebo *addWall* pro přidání vnitřní zdi. Tato metoda se zavolá nad aktivním patrem budovy. Jako koncový bod zdi slouží bod získaný kliknutím v případě zapnutého přichytávání posunutý na patřičnou pozici. U obvodové zdi se koncový bod nastaví jako počáteční bod další zdi a pokračuje se v kreslení. Tato funkce zajišťuje plynulé navazování jednotlivých obvodových zdí.

### 6.4.2 Dveře a okna

Při kreslení dveří nebo okna se při pohybu myší nad kreslicí oblastí vytvoří dočasný objekt, který se vykresluje na pozici kurzoru. Při pohybu myši se volá metoda *selectObject* u aktivního patra budovy, která vrací objekt v blízkosti kurzoru. Pokud je jako objekt vrácena zeď, tak se pomocí metod getRotation, která vrátí natočení nejbližší zdi, a snapToLine, které přichytí bod k přímce dané dvěma body, umístí na tuto zeď. Při stisknutí levého tlačítka myši se zavolá metoda *addWindow* nebo *addDoors* nad aktivním patrem budovy, která vybere patřičnou zeď a pomocí metody addWindow nebo addDoors přidá do seznamu objektů umístěných na této zdi nový objekt.

### 6.4.3 Schodiště

Při vkládání schodiště se při pohybu myší nad kreslicí oblastí vytvoří dočasný objekt, jenž se vykresluje na pozici kurzoru. Při kliknutí se u aktivního patra zavolá metoda *addUpStairs*, která do seznamu schodišť do vyššího patra přidá nový objekt.

### 6.4.4 Podlaží

Při vkládání nového patra se zavolá metoda *addFloor*, která do seznamu pater budovy přidá nové patro. Do tohoto patra se z předchozího patra zkopírují odkazy na všechna schodiště vedoucí do vyššího patra a zde se vloží do seznamu pater vedoucích do nižšího patra. Díky tomu je zajištěno, že když se dané schodiště v jednom z pater smaže, nebo se změní některá jeho vlastnost, bude změna patrná v obou patrech. Dále se z předchozího patra překopírují obvodové zdi. Ty se zde ovšem vytvoří jako nové objekty, což umožňuje různý tvar obvodu jednotlivých pater.

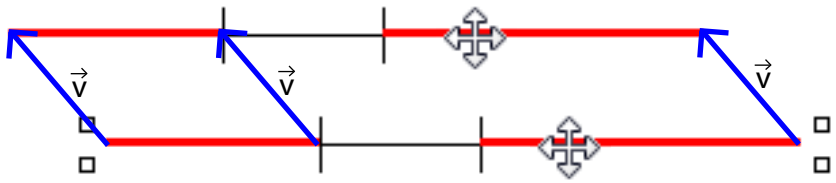
## 6.5 Upravování objektů

Pro upravení objektu je potřeba nejdříve daný objekt vybrat pomocí nástroje kurzoru a kliknutí na objekt. Nebo pomocí stisknutí tlačítka a tažením myši se vytvoří obdélníkový výběr více objektů. V tomto případě je možno objekty pouze pohybovat a mazat je. Pro zjištění požadované akce slouží metoda *getTool*, která vrací patřičný nástroj u vybraného objektu (pohyb, změna velikosti).

### 6.5.1 Pohyb

Pro pohyb objekty slouží metoda *Move*. Tato metoda je implementovaná v třídě AbstractWall, protože je společná pro zeď a obvodovou zeď, dále v třídě OnePointGraphicObject pro okno a dveře a v třídě Stairs pro schodiště. Metoda přijímá jako parametr vektor pohybu. Vektor pohybu se při pohybu myši určí z předchozí pozice myši k nové pozici.

Při pohybu zdí se uloží souřadnice počátku a konce zdi. Poté se k těmto souřadnicím přičte vektor posunutí, následně se provede test kolize s ostatními objekty. V případě kolize se vrátí zpět staré souřadnice. Pokud ke kolizi nedošlo, projde se seznam všech objektů, které patří k dané zdi, a k jejich počáteční souřadnici se opět přičte vektor posunutí. Posun zdi je ukázán na obrázku 6.5




Obrázek 6.5: Pohyb zdí.

Při pohybu oknem nebo dveřmi se uloží souřadnice počátku. Následně se vytvoří nový bod, který vznikne sečtením počátečního bodu objektu a vektoru posunutí. Tento bod se pomocí funkce snapToLine přichytí k přímce, na které leží zeď, po které tento objekt posouváme. Na tento bod se nastaví souřadnice počátku posouvaného objektu a provede se kolizní test s ostatními objekty ve zdi. V případě kolize se nastaví počáteční bod zpět na původní.

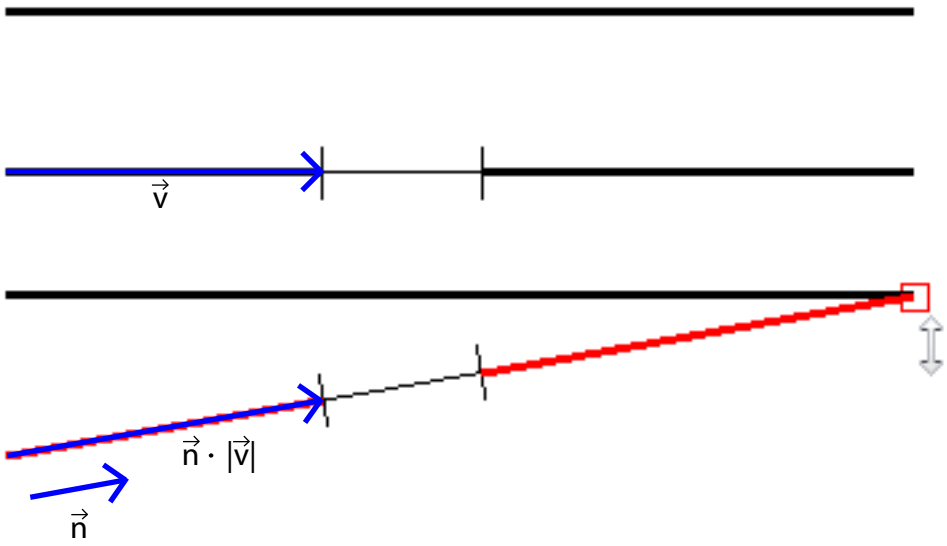
Při pohybu schodištěm se pouze k počátečnímu bodu schodiště přičte vektor posunutí.

### 6.5.2 Změna velikosti

Pro změnu velikosti objektu slouží metoda *Resize*. Tato metoda je implementovaná v třídě AbstractWall, protože je společná pro zeď a obvodovou zeď, dále v třídě OnePointGraphicObject pro okno a dveře. Metodě se jako parametr předává bod poslední pozice myši. Dále se u objektu nastaví vlastnost click, která udává, jestli se upravuje počátek (hodnota Begin) nebo konec objektu (hodnota End).

Při změně velikosti zdi je možné měnit velikost zdi a zároveň pozici počátečního nebo koncového bodu. V případě, že je povoleno přichytávání k objektům, se nejdříve cílový bod pokusí přichytit k nejbližší zdi. Poté se počáteční nebo koncový bod nastaví na cílový bod a provede se kolizní test. Pokud nastane kolize, vrátí se posunutí zpět. V případě, že kolize nenastane, se musí posunout všechny objekty ležící na zdi. Nejdříve se vypočte nový úhel natočení zdi a jednotkový vektor z počátku do konce zdi. Poté se pro každý objekt určí jeho

vzdálenost od počátku nebo konce zdi. Nový počáteční bod se určí jako součet počátečního nebo koncového bodu s jednotkovým vektorem zdi, který je vynásoben vzdáleností objektu od počátku nebo konce zdi. Objektu se nastaví nově vypočtený počáteční bod a úhel natočení. Princip je ukázán na obrázku 6.6.




Obrázek 6.6: Změna velikosti zdi.

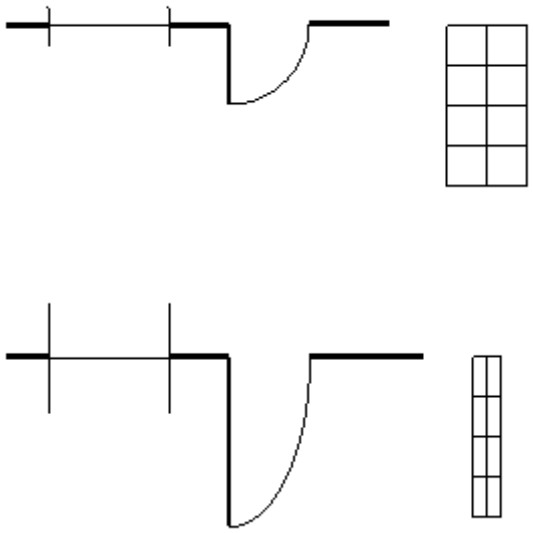
U změny velikosti okna nebo dveří se mění pouze jejich velikost. Nejdříve se cílový bod přichytí k přímce, na které leží rodičovská zeď objektu, pomocí metody snapToLine. Pokud se mění počátek objektu, tak se určí vektor z přichyceného bodu do koncového bodu objektu. Nová velikost objektu se nastaví na velikost tohoto vektoru. Nakonec se koncový bod nastaví zpět na původní, tím se zajistí, že se nepohne koncový bod, ale počáteční. Při změně koncového bodu se určí vektor z přichyceného bodu do počátečního a velikost objektu se nastaví na velikost tohoto vektoru. Po těchto změnách se provede kolizní test. Pokud dojde ke kolizi, vrátí se souřadnice a velikost zpět na původní.

U oken, dveří a schodiště je implementována metoda *resizeSide*, která jako parametr přijímá vektor pohybu kurzoru. Tato metoda umožňuje měnit šířku těchto objektů. Metoda určí velikost posunutí vůči vybranému objektu a na základě toho se mění vlastnost *size* objektu. Změna šířky objektu je ukázána na obrázku 6.7.

### 6.5.3 Mazání

Pro smazání objektu je u každého vytvořeného objektu uložena reference na rodičovský objekt. U dveří a oken je to zeď, u zdí a schodiště patro. Pro odebrání objektu slouží metoda *Remove*, která daný objekt odebere z příslušného seznamu objektů rodičovského objektu. Při zmáčknutí klávesy *Delete* se u vybraných objektů zavolá metoda Remove a tím dojde k odstranění objektů.

Pro odstranění aktivního patra slouží tlačítko v panelu nástrojů. Po jeho stisknutí se zobrazí dialogové okno pro potvrzení smazání celého patra všech objektů v něm. Po potvrzení se zavolá metoda *deleteFloor* třídy Building, která odstraní aktivní patro ze seznamu pater. Po smazání se jako aktivní patro nastaví předchozí. Pokud předchozí patro neexistuje, posune se aktivita na patro vyšší. Pokud neexistuje ani to a mazané patro je poslední v budově, vytvoří se nové prázdné patro.



Obrázek 6.7: Nahoře původní objekty, dole objekty se změněnou šířkou.

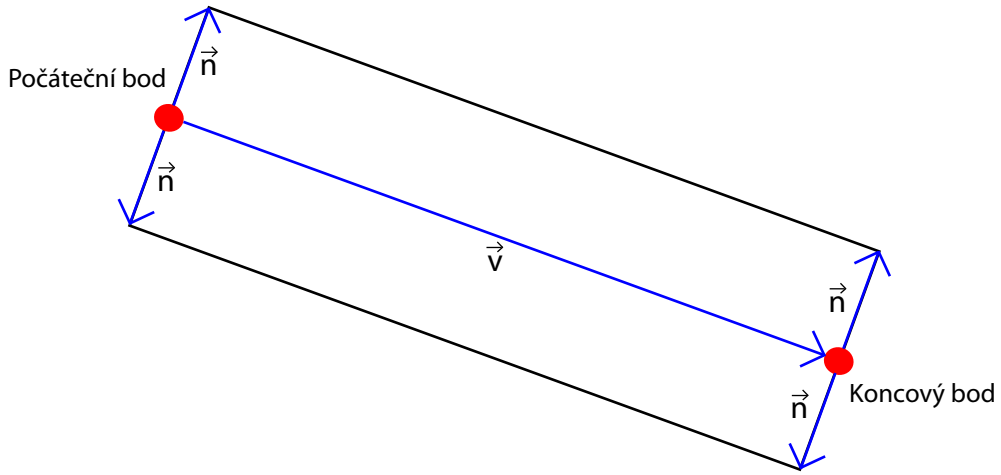
## 6.6 Přichytávání objektů a kolize

### 6.6.1 Výběr objektů

Výběr objektu je možné provést kliknutím v blízkosti objektu nebo podržením levého tlačítka myši a táhnutím. V tomto případě se vytvoří obdélníkový výběr, kde se vyberou všechny objekty ležící uvnitř tohoto obdélníku. Pro výběr objektů musí být zvolen nástroj kurzoru. Pro výběr objektu je ve třídě Floor definována metoda *selectObject*, která přijímá jako argument bod, ve kterém se kliklo při výběru, nebo bod levého horního rohu a pravého dolního rohu obdélníkového výběru. Pokud se vybere jeden objekt je uložen ve vlastnosti *lastSelected* třídy Editor. Seznam více objektů je uložen ve vlastnosti *SelectedObjects*. U všech takto vybraných objektů se nastaví metoda selected na hodnotu true. Pokud před výběrem byly vybrány jiné objekty, nastaví se u nich tato metoda zpět na hodnotu false. Tato vlastnost způsobí označení objektu a umožní ho upravovat.

Při výběru jednoho objektu se postupně prochází seznam všech objektů v patře a volá se u nich metoda Contains. Tato metoda určí, jestli se objekt nachází v blízkosti kliknutí. Tato metoda je implementovaná ve třídách AbstractWall, Window, Doors a Stairs. V této metodě se vytvoří obdélník kolem objektu, jehož velikost závisí na vlastnosti precision třídy Editor. Následně se souřadnice tohoto obdélníku předají metodě Colision, která vyhodnotí, jestli bod leží uvnitř tohoto obdélníku. Pokud se takto narazí na zeď, volá se opět metoda selectObject u této zdi a prochází se stejným způsobem seznam objektů na této zdi. Pro určení obdélníku kolem objektu se vypočte vektor z počátečního bodu do koncového. Potom se z tohoto vektoru vytvoří kolmý vektor a vynásobí se potřebným posunutím. Tento vektor se potom přičte i odečte od počátečního i koncového bodu, čímž dostaneme čtyři body obdélníku kolem objektu. Princip výběru objektu ukazuje obrázek 6.8.

Při výběru více objektů se prochází seznam všech objektů v patře a volá se u nich metoda Contains, jíž se předávají jako parametry dva body určující obdélník výběru. Pokud




Obrázek 6.8: Výběr objektu.

se celý objekt nachází uvnitř obdélníku, přidá se do seznamu vybraných objektů. Pokud výběr neobsahuje žádnou zeď, projdou se i seznamy objektů v jednotlivých zdích a pokusí se vybrat tyto. Tato metoda Contains je implementována ve třídě GraphicObject. Metoda kontroluje, jestli souřadnice počátečního a koncového bodu leží uvnitř daného obdélníku. Pokud oba body leží uvnitř obdélníku, musí jejich souřadnice být větší než souřadnice levého horního rohu a menší než souřadnice pravého dolního rohu obdélníku výběru.

### 6.6.2 Přichytávání objektů

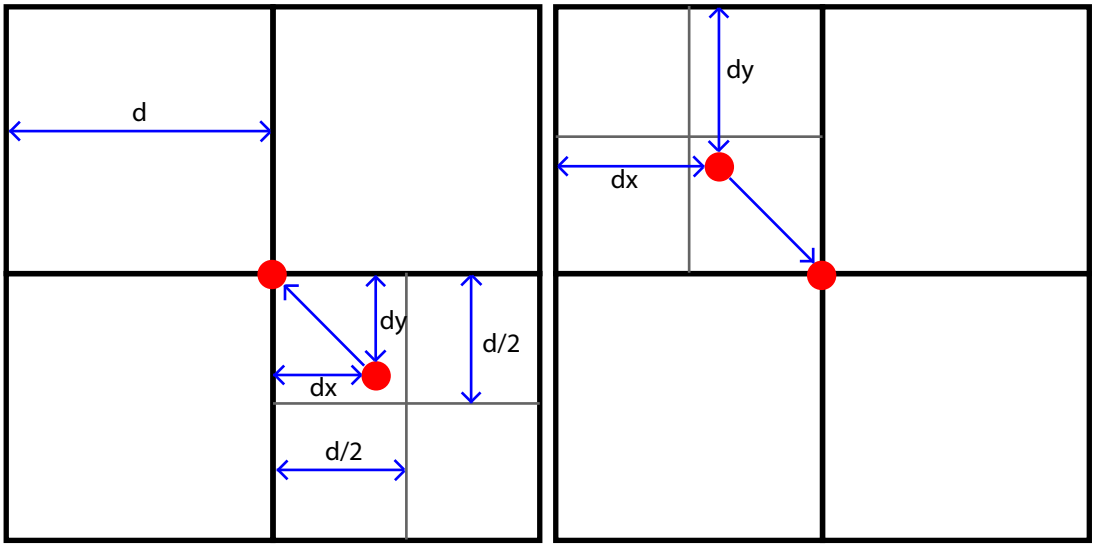
Zdi je možno přichytávat k zobrazené mřížce nebo k ostatním zdím. Okna a dveře se přichytávají ke zdím. Vzdálenost, do které se objekty vzájemně přichytávají, je dána vlastností *precision* třídy Editor.

Při přichytávání zdí k mřížce se používá metoda *SnapToGrid* v třídě Editor, které se jako parametr předá bod, jenž se má přichytit k mřížce. Této metodě se postupně předá počáteční a koncový bod zdi zjištěný od uživatele kliknutím do kreslicí oblasti. Nová zeď se vytvoří z těchto přichycených bodů. Při přichycení k mřížce se pomocí operace modulo ujistí odchylka bodu od bodu mřížky. Pokud je tato odchylka menší než polovina velikosti mřížky, odečte se tato odchylka od bodu a tím se přichytí k předcházejícímu bodu mřížky. Pokud je vetší, přičte se zbývající část do následujícího bodu mřížky viz. obrázek 6.9.

Při přichytávání zdí k sobě, při vytváření, nebo upravování se pomocí metody *selectWall* u aktivního patra budovy zkusí vybrat zeď u daného bodu. Pokud se zeď najde, tak se vypočtou vektory z počátečního a koncového bodu zdi do přichytávaného bodu. Pokud je délka některého z vektorů menší než nastavená přesnost přichytávání, přichytí se bod k počátku nebo konci zdi. V případě, že se bod nepřichytí k okrajovým bodům zdi, tak se pomocí metody snapToLine přichytí k přímce, na které leží zeď.

### 6.6.3 Pravoúhlý mód

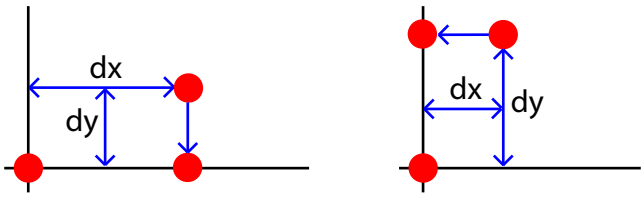
Pro usnadnění tvorby pravoúhlých objektů byla přidána možnost zapnutí pravoúhlého módu. Ten se aktivuje při stisku klávesy *Ctrl* nebo zapnutím příslušné volby v hlavním menu aplikace. Tento mód funguje při vkládání zdí. Pro tyto účely byla implementována




Obrázek 6.9: Přichycení bodu k mřížce.

metoda *ort* ve třídě Editor.

První přetížení této metody přebírá jako parametr počáteční a koncový bod tvořené zdi. V této metodě se určí rozdíl mezi jednotlivými souřadnicemi těchto dvou bodů. Pokud je větší rozdíl x-ových souřadnic, nastaví se u koncového bodu y-ová souřadnice stejná jako u počátečního. V případě, že je větší rozdíl y-ových souřadnic, nastaví se u koncového bodu y-ová souřadnice stejná jako u počátečního. Princip ukazuje obrázek 6.10.

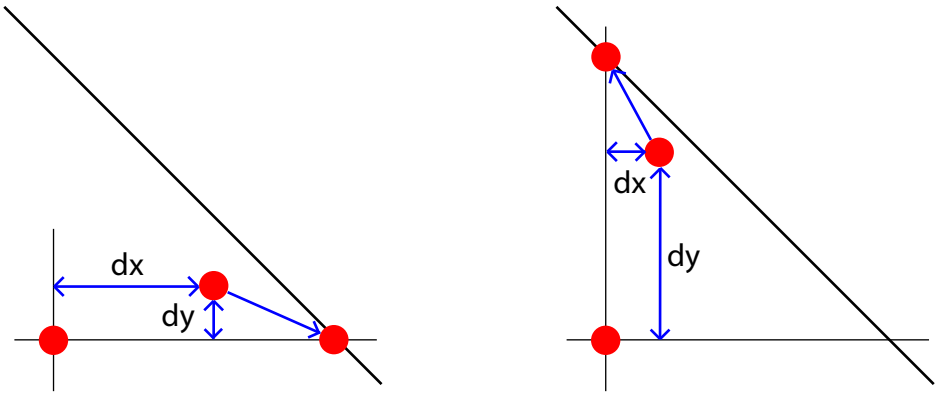



Obrázek 6.10: Pravoúhlé přichycení bodu.

Druhé přetížení metody ort přebírá jako parametr počáteční bod koncový bod a zeď, ke které se má koncový bod pravoúhle přichytit. V této metodě se určí jednotlivé složky ve směrnicovém tvaru přímky, na které leží cílová zeď. Určí se rozdíly souřadnic mezi počátečním a koncovým bodem. Pokud je větší rozdíl x-ových souřadnic, dopočítá se ze směrnicového tvaru přímky x-ová souřadnice pro y-ovou souřadnici počátečního bodu. Jestli je větší rozdíl y-ových souřadnic, dopočítá se y-ová souřadnice. Pokud je x-ová souřadnice počátku a konce cílové zdi stejná, nastaví se u přichyceného bodu x-ová souřadnice na x-ovou souřadnici počátku zdi a y-ová souřadnice na y-ovou souřadnici počátečního bodu. Přichycení je ukázáno na obrázku 6.11.

### 6.6.4 Kolize

Testy kolizí jednotlivých objektů zabraňují křížení zdí a překrývání objektů, které se do zdí vkládají.



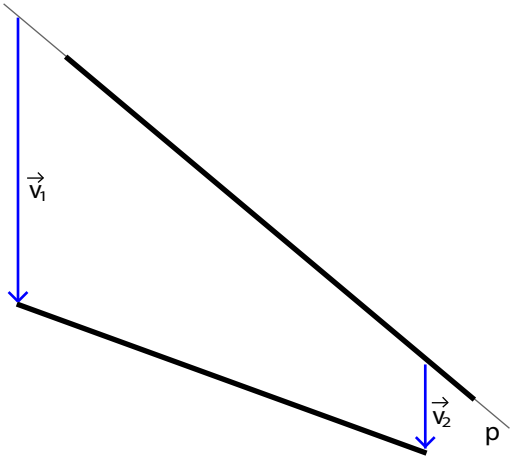

Obrázek 6.11: Pravoúhlé přichycení bodu ke zdi.

Pro test kolizí mezi zdmi je implementována metoda *wallTest* ve třídě Floor a jako parametr přijímá zeď, pro kterou se určuje kolize s ostatními. Tato metoda se využívá při vkládaní, upravování a pohybu zdi. Metoda prochází seznam všech zdí v patře. Nejdříve se otestují souřadnice krajních bodů obou zdí a pokud se nepřekrývají, kolize nenastává viz obrázek 6.12. Pokud předchozí test kolizi nevyloučí, určí se úhel mezí těmito dvěma zdmi. Pokud leží na stejné přímce, pomocí metody isOnLine se otestuje pro jednotlivé krajní body zdí jestli se nepřekrývají. Jestliže na jedné přímce neleží, tak se pomocí metody snapToLine přichytí okrajové body jedné zdi k druhé a pak se určí vektory z okrajových bodů do těchto přichycených bodů. Pokud oba vektory směřují vlevo nebo oba vpravo od přímky, na které leží zeď, ke kolizi nedochází jak je ukázáno na obrázku 6.13. Jestliže vektory směřují různými směry od přímky, určí se vektory z počátečního bodu jedné zdi do obou krajních bodů druhé zdi, stejně tak z koncového bodu zdi do obou krajních bodů druhé zdi viz. obrázek 6.14. V případě, že je úhel mezi těmito dvěma vektory větší než 180°, došlo ke kolizi. Tento test zamezuje křížení jednotlivých zdí.

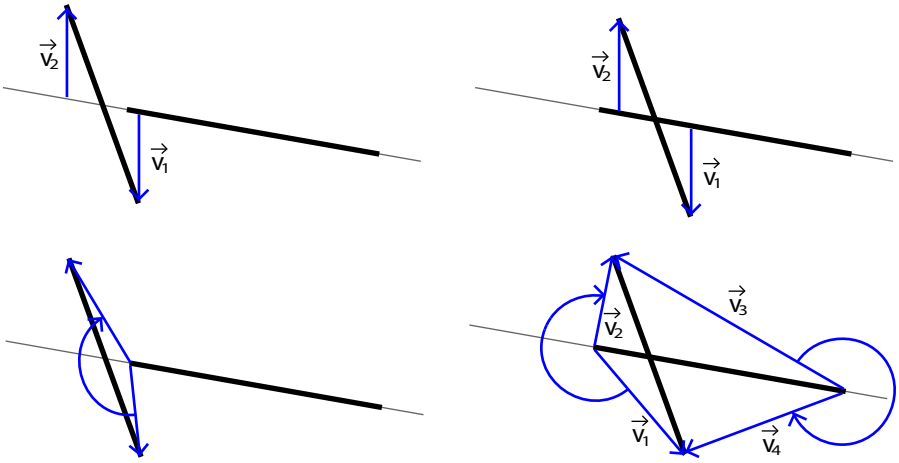
Obrázek 6.12: Vyloučení kolize testem souřadnic.

## 6.7 Práce se soubory

Pro uložení vytvořeného plánku do souboru slouží metoda *toXml* deklarovaná v třídě GraphicObject. Tato metoda vrací vytvořený XML uzel. Při ukládání plánku do souboru se nejdříve zavolá tato metoda u třídy Building. Zde se vytvoří hlavní uzel XML dokumentu a přidají se do něj uzly obsahující nastavení editoru. Následně se projde seznam všech pater v budově a nad každým se zavolá metoda toXML. Pro každé patro tato metoda vrátí uzel, který představuje dané patro a připojí se k hlavnímu uzlu. V patře se projdou seznamy všech objektů v patře a opět se u nich zavolá metoda to XML a takto postupně u všech objektů, které obsahují další objekty. Tímto se seskládá celý strom XML dokumentu. Fyzické objekty jako jsou zdi, okna, dveře a schodiště vytvářejí XML uzel, který obsahuje všechny informace potřebné k znovuvytvoření objektu při načítání. Takto vytvořený dokument se




Obrázek 6.13: Test různoběžných zdí.




Obrázek 6.14: Test různoběžných zdí.

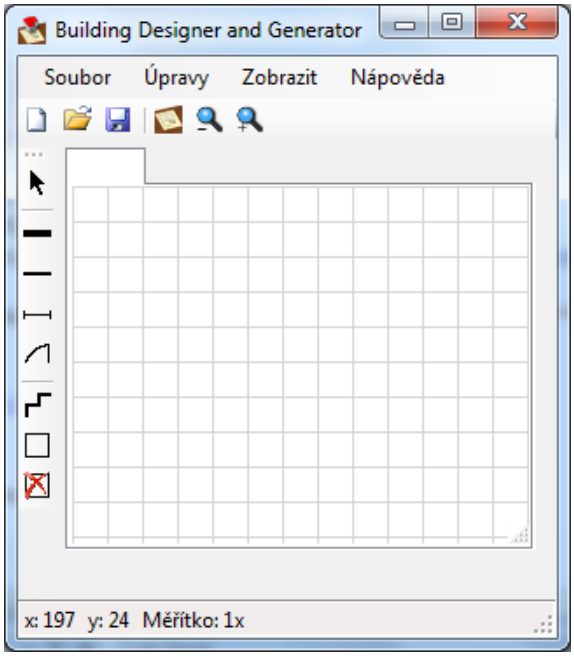
zapíše do cílového souboru. Před vlastním uložením se v metodě checkSave zkontroluje, jestli se soubor od posledního uložení změnil, případně jestli je zadáno jméno souboru.

Pro načtení plánku ze souboru slouží statická metoda *load* ve třídě Fce. Tato metoda nejdříve načte XML z daného souboru. Poté se vytvoří nová instance třídy building a začne se procházet XML dokument po jednotlivých elementech představujících patra. Uvnitř těchto elementů se prochází jednotlivé objekty v patře a u zdí objekty v nich. Pokud se narazí na některý element, který představuje fyzický objekt, vytvoří se z uložených informací objekt nový. Tímto způsobem se postupně znovuvytvoří všechny objekty v budově. Nakonec se načtou jednotlivá nastavení editoru.

Export jednotlivých částí budovy do rastrového obrázku probíhá v metodě, jenž reaguje na událost kliknutí na tuto možnost v menu. Při exportu se vytvoří nová instance třídy Bitmap reprezentující bitmapový obrázek. Potom se pomocí statické metody FromImage třídy Graphics získá její instance, která umožňuje kreslit do tohoto obrázku. Poté se zavolá metoda Paint2D u aktivního patra budovy a jako parametr se jí předá takto vytvořená třída Graphics. Tímto se vykreslí budova do obrázku. Potom se obrázek uloží do cílového souboru ve zvoleném formátu.

## 6.8 Uživatelské rozhraní

Hlavní okno aplikace představuje třída *MainWindow*, která rozšiřuje třídu Form z knihovny .NET Frameworku. Jsou v ní definovány metody, které reagují na akce uživatele jako je kliknutí na položku v menu, ukládání souboru a podobně. Vzhled hlavního okna aplikace je ukázán na obrázku 6.15.

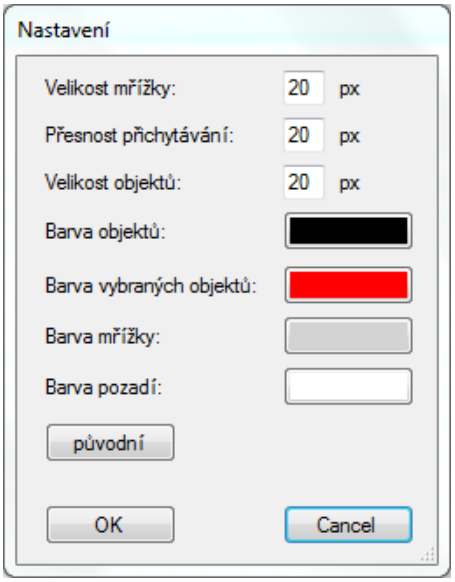



Obrázek 6.15: Hlavní okno aplikace.

V případě, že uživatel klikne na možnost nastavení aplikace v hlavním menu, zobrazí se pomocí metody *showDialog* dialogové okno tvořené třídou Options ukázané na obrázku 6.16. Dialogové okno pro nastavení aplikace umožňuje nastavení základních vlastností editoru jako barvy jednotlivých vykreslovaných částí, vzdálenost, do které se objekty budou přichytávat, velikost mřížky a velikost vytvářených objektů. Po nastavení hodnot a potvrzení stisknutím tlačítka OK se provede kontrola platnosti nových parametrů. Pokud jsou platné, nastaví se jednotlivé vlastnosti na dané hodnoty. Pokud jsou neplatné, zobrazí se informační dialog o zadání neplatných údajů a žádná změna se neprovede. Tato nastavení jsou spravována pomocí objektu *Properties.Settings.Default*. Tento objekt zajistí uložení nastavení a jeho načtení při spuštění aplikace.

V hlavním okně aplikace můžeme v horní části nalézt menu pro přístup k jednotlivým funkcím editoru a dále dva panely nástrojů. Horní z nich slouží pro přístup k nejčastějším funkcím jako je uložení nebo otevření nového souboru. Druhý panel nástrojů slouží pro výběr nástroje v editoru. Největší plochu okna zabírá plátno pro kreslení tvořené třídou *EditTabControl*. Ve spodní části okna se nachází stavový řádek, který zobrazuje informaci o pozici kurzoru, aktuálním měřítku editoru a vybraném nástroji. Jednotlivé časti jsou ukázány na obrázku 6.17.

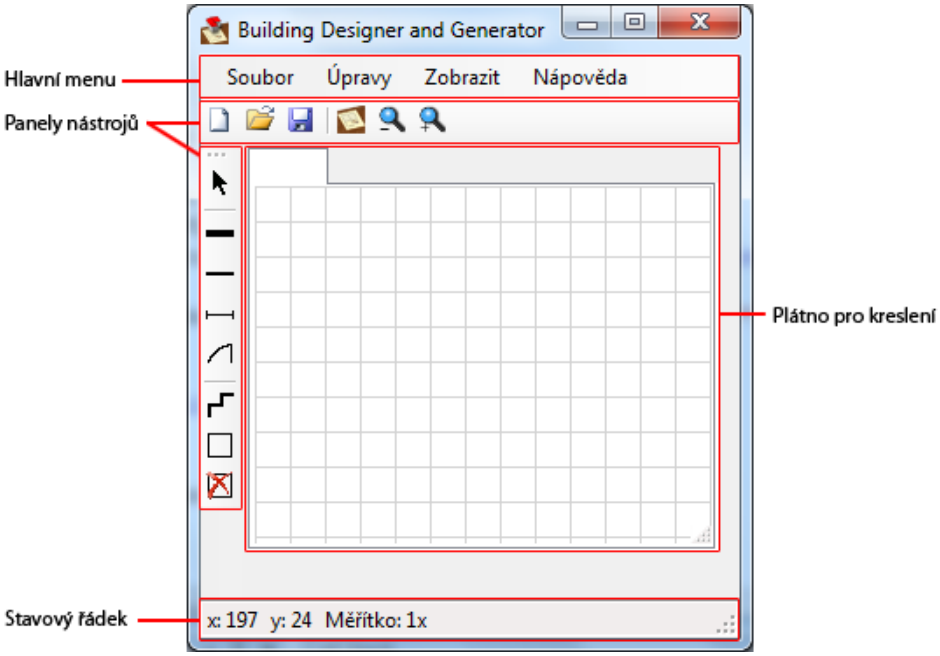
Pro výběr nástroje slouží panel nástrojů na levé straně aplikace. Pokud uživatel nástroj vybere klepnutím levého tlačítka myši na dané tlačítko, nastaví metoda, která obsluhuje tuto událost, ve třídě Editor vlastnost *tool* na patřičný nástroj. V případě tlačítka pro odstranění patra se zobrazí pomocí metody Show třídy MessageBox definované v .NET



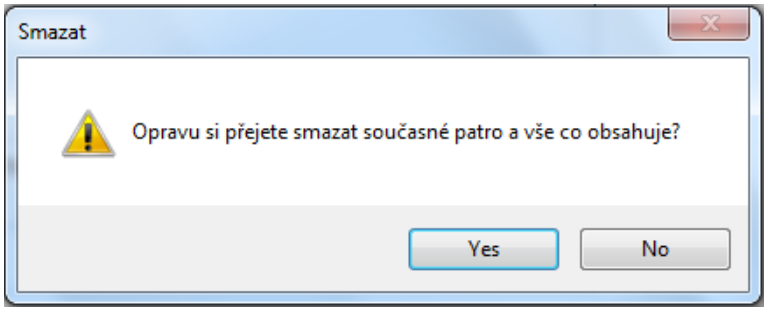

Obrázek 6.16: Nastavení aplikace.

Frameworku dialogové okno pro potvrzení volby viz obrázek 6.18. Poté se zavolá metoda deleteFloor třídy Building pro odstranění patra a metoda deleteTab, která odstraní záložku pro dané patro.

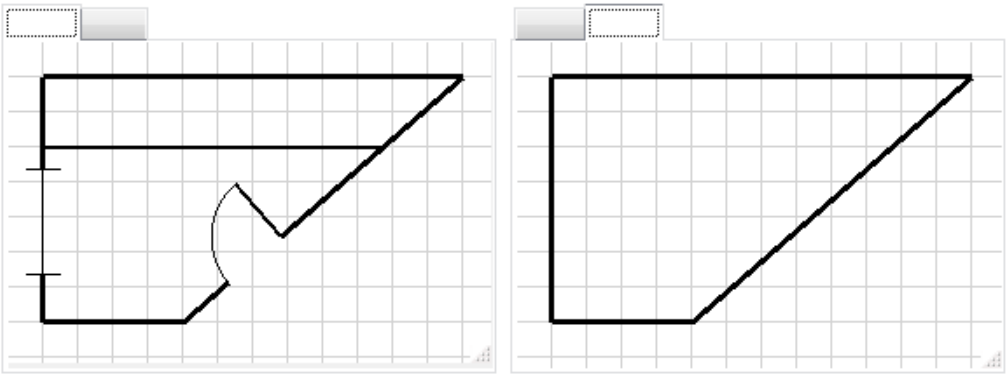
Jako kreslicí plátno slouží třída *ResizableTabPage*, která rozšiřuje třídu tabPage o možnost změny velikosti. Pro každé patro budovy je vytvořena jedna instance této třídy. Všechny tyto instance spravuje třída EditTabControl, která umožňuje přepínání mezi jednotlivými patry. Tato třída rozšiřuje třídu TabControl pro možnost práce s třídami ResizableTabPage. Tyto třídy jsou ukázány na obrázku 6.19.




Obrázek 6.17: Jednotlivé části hlavního okna.




Obrázek 6.18: Informační dialog.

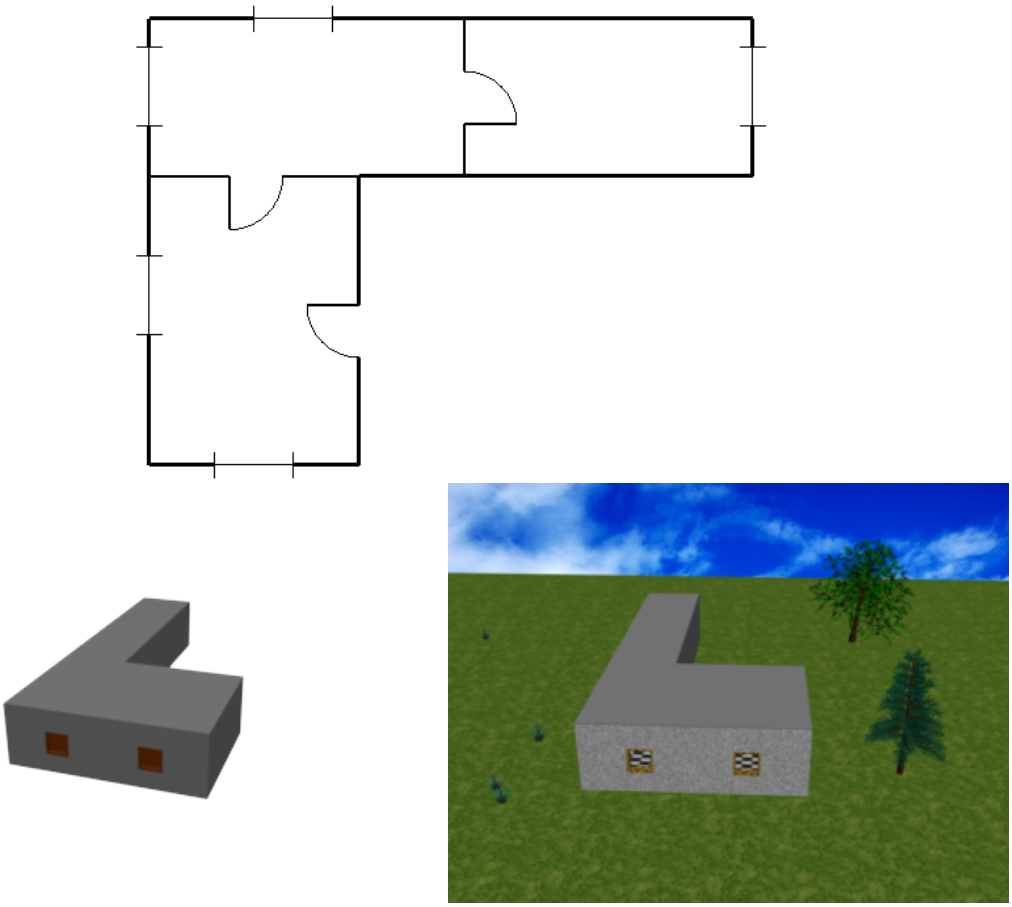


Obrázek 6.19: Plátno pro kreslení.

# Kapitola 7

# Generování 3D modelu

Při vygenerování 3D modelu je nejdříve zapotřebí sestavit scénu z objektů vytvořených v rámci 2D plánku. Tuto scénu je zapotřebí uložit do zvoleného 3D formátu. Tento základní model bude možné při zvolení vhodného formátu upravit v některém pokročilém 3D editoru. Příklad procesu tvorby výsledného modelu je ukázán na obrázku 7.1.



Obrázek 7.1: Nahoře 2D plánek, vlevo dole návrh 3D modelu vygenerovaného editorem, vpravo dole vygenerovaný 3D model upravený ve vhodném 3D editoru.

## 7.1 Návrh řešení

Nejdříve by bylo zapotřebí v nějakém programu pro 3D modelování například Blender nebo 3Ds Max vytvořit modely pro jednotlivé grafické objekty jako okna, dveře a schodiště. Pro zdi bude zapotřebí vytvářet model dynamicky pro možnost vložení dveří a oken do zdi.

Pro generování modelu by bylo vhodné rozšířit rozhraní Render o metodu pro vykreslení 3D modelu budovy. V této metodě by se postupně sestavoval výsledný graf scény. Při vykreslení celé budovy by se postupně tato metoda volala na jednotlivá patra budovy. V patře by se tato metoda zavolala u všech objektů, které do něj patří. Stejně tak u zdí by se tato metoda zavolala u všech objektů vložených do zdi. V případě oken, dveří a schodišť by se v této metodě do grafu scény přidaly transformace pro umístění objektu na dané místo. Za tyto transformace by se vložil již vytvořený model daného objektu. Model zdi by bylo možné vytvořit pomocí konstruktivní geometrie, kdy by se v kvádru tvořícím zeď pomocí rozdílového operátoru vytvořil prostor pro vložení modelu okna nebo dveří. Do výsledné scény by bylo nutné vložit kameru a osvětlení.

Pro nastavení vytváření toho modelu by bylo nutné rozšířit stávající uživatelské rozhraní o volby pro nastavení parametrů modelů jednotlivých objektů jako je například výběr textury pokrývající objekt nebo výška objektů. Dále by bylo vhodné u jednotlivých objektů dovolit náhled 3D modelu před jeho vlastním vytvořením.

Vygenerovaný model by bylo možné uložit ve zvoleném 3D formátu pro prohlížení v externí aplikaci. Do aplikace by však měl být vložen interní prohlížeč tohoto modelu. Pro jeho implementaci by bylo možné využít například XNA Framework, který zapouzdřuje práci s DirectX pro použití v prostředí .NET.

## 7.2 Formát 3D modelu

Pro vygenerování 3D modelu je zapotřebí vybrat vhodný formát, který umožní jednoduché vytvoření celé scény a její export do souboru, který bude možné otevřít v jiné aplikaci. Do tohoto formátu by mělo být možné exportovat předvytvořené modely jednotlivých objektů z některého 3D modelovacího programu. V této části je uvedeno několik nejznámějších formátů.

### 7.2.1 VRML

VRML je deklarativní jazyk, který slouží pro popis 3D scén. Formát byl původně vytvořen pro přenos 3D dat po síti. Formát souborů ve formátu VRML je založen na textovém popisu scény. Celá scéna je zde upořádána do stromové struktury, do které se přidávají jednotlivé transformace a objekty. Objekty jsou zde popsány pomocí hraniční reprezentace. Lze u nich definovat texturu, která tento objekt pokrývá. Do scény je možné vložit další objekty jako světla a kamery. [13]

### 7.2.2 X3D

Tento formát vychází z jazyka VRML, který rozšiřuje o nové datové objekty a možnost uložení v XML reprezentaci. Formát dat je založen na jazyce XML. Prozatím je ovšem tento formát poměrně málo rozšířen a podporován. [8]

### 7.2.3 COLLADA

Tento formát pro reprezentaci 3D scény je založen na formátu XML. Formát byl původně vytvořen pro přenos 3D dat mezi různými aplikacemi. Tento formát je definovaný pomocí XML schématu, jež obsahuje definici základních typů objektů, které je možné vložit do scény. Do scény je možné kromě vložení geometrie těles vložit také animace a fyziku. [3]

# Kapitola 8

# Závěr

Při tvorbě aplikace jsem si nejdříve stanovil cíle projektu v podobě požadovaných funkcí poskytovaných editorem. Po stanovení cílů jsem navrhl objektový model aplikace s ohledem na maximální využití již hotových tříd v .NET Frameworku. Tento model se ovšem v průběhu vlastní implementace aplikace ještě musel upravovat. Dále jsem vybral formát dat, se kterými aplikace pracuje a navrhl jednoduché grafické rozhraní celé aplikace. Po návrhu aplikace jsem přistoupil k vlastní implementaci celé aplikace. Dále jsem diskutoval možnost generování 3D modelu budovy. Tímto jsem splnil zadání celého projektu.

Aplikace poskytuje základní avšak dostačující funkčnost pro vytváření orientačních plánků budov, v budoucnu by ovšem bylo vhodné aplikaci rozšířit o následující funkce. Kromě již diskutovaného generování 3D modelu by bylo vhodné aplikaci rozšířit o knihovnu objektů. Do této knihovny by se načítaly nové objekty například z xml souboru, který by obsahoval jejich popis a vektorovou reprezentaci 2D modelu případně popis 3D modelu objektu. Takto načtené objekty by bylo možné vkládat do plánku. Jednalo by se například o možnost vkládání nábytku do jednotlivých místností.

Dalším vhodným rozšířením by byla detekce hran nebo rohů v načtené předloze plánku. Při detekci hran by bylo možné při rozeznání zdi v předloze přímo vytvořit nový grafický objekt. V případě detekce rohů by bylo možné přichytávání vytvářených objektů k těmto rozpoznaným bodům.

# Literatura


[1] 3ds Max - 3D Modeling, Animation, and Rendering Software - Autodesk. [online], [cit. 2010-04-12].
URL
http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13567410

[2] ArchiCAD 13 - Overview. [online], [cit. 2010-04-12].
URL http://www.graphisoft.com/products/archicad/

[3] COLLADA - Digital Asset Exchange Schema for Interactive 3D. [online], [cit. 2010-04-14].
URL http://www.khronos.org/collada/

[4] Microsoft Office Visio 2007. [online], [cit. 2010-04-12].
URL http://www.microsoft.com/cze/office/programs/visio/highlights.mspx

[5] MSDN Library. [online], [cit. 2010-04-20].
URL http://msdn.microsoft.com/hi-in/library/ms123401.aspx

[6] SmartDraw - Communicate Visually. [online], [cit. 2010-04-12].
URL http://www.smartdraw.com/

[7] Usability basics. [online], [cit. 2010-03-29].
URL http://www.usability.gov/basics/index.html

[8] X3D Specifications. [online], [cit. 2010-04-14].
URL http://www.web3d.org/x3d/specifications/x3d/

[9] XML Essentials. [online], [cit. 2010-04-04].
URL http://www.w3.org/standards/xml/core

[10] KRŠEK, P.: *Základy počítačové grafiky IZG Studijní opora.*

[11] LANCASTER, D.: Nonlinear Graphic Transforms. [online], [cit. 2010-04-04].
URL http://www.tinaja.com/glib/nonlingr.pdf

[12] NAGEL, C.; et al.: *C# 2008 Programujeme profesionálně.* Computer Press, první vydání, 2009, ISBN 978-80-251-2401-7.

[13] ŽÁRA, J.; et al.: *Moderní počítačová grafika.* Computer Press, druhé vydání, 2004, ISBN 80-251-0454-0.

[14] STONE, D.; et al.: *User Interface Design and Evaluation.* ELSEVIER, 2005, ISBN 0-12-088436-4.

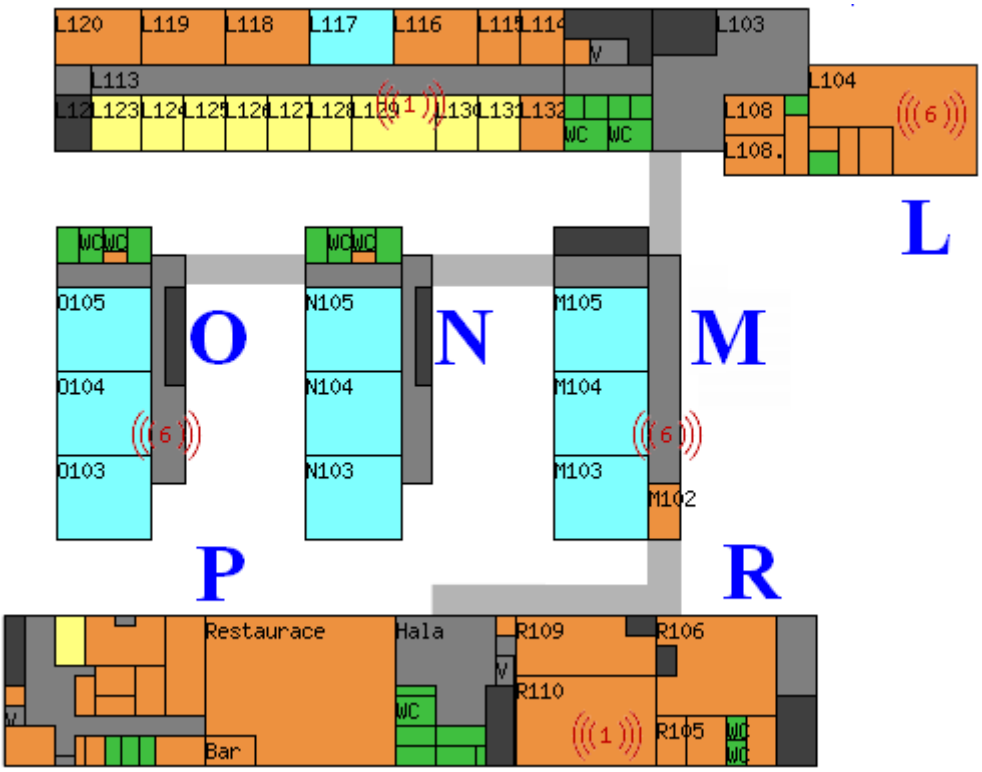
# Příloha A

# Obsah CD

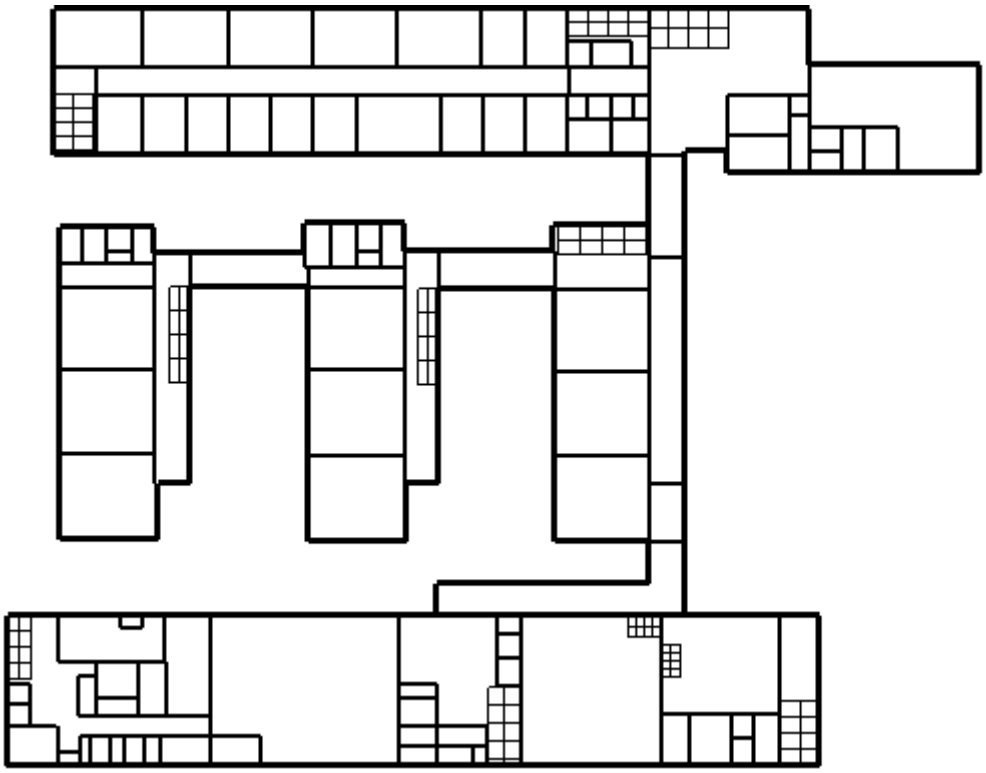
`bp` adresář obsahuje zprávu ve formátu pdf
`edit` adresář obsahuje zdrojové soubory práce pro LaTeX
`poster` adresář obsahuje plakátek prezentující práci
`source` adresář obsahuje zdrojové kódy celé aplikace
`binary` adresář obsahuje spustitelnou verzi aplikace
`documentation` adresář obsahuje programovou dokumentaci ve formátu pdf
`example` adresář obsahuje ukázkový plánek vytvořený v aplikaci

# Příloha B

# Ukázka vytvořeného plánu




Obrázek B.1: Předloha.



Obrázek B.2: Vytvořený plán.