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

VYHLEDÁVÁNÍ TRASY V MAPĚ BUDOVY

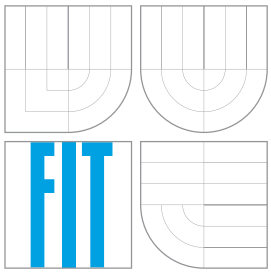
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ANTONÍN MARKO

BRNO 2016

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

VYHLEDÁVÁNÍ TRASY V MAPĚ BUDOVY

PATHFINDING IN A BUILDING MAP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ANTONÍN MARKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ZACHARIÁŠ

BRNO 2016

Abstrakt

Většina velkých budov obsahuje statické panely s plánkem budovy, které nejsou přehledné a trvá dlouhou dobu najít konkrétní místnost a cestu k ní. Předmětem této práce je návrh a implementace aplikace pro zobrazení mapy budovy ve 3D a vyhledávání trasy v ní.

Abstract

The majority of large buildings contains static public panels with a building map, which are not well-arranged and it takes long time to find concrete area and find path to it. Subjects of the thesis are design and implementation of an application for 3D demonstration of a building's map and pathfinding in the map.

Klíčová slova

WPF, C#, MVVM, Vyhledávání trasy v budově, 3D mapa

Keywords

WPF, C#, MVVM, Pathfinding in a building, 3D map, in-door, pathfinding

Citace

Antonín Marko: Vyhledávání trasy v mapě budovy, bakalářská práce, Brno, FIT VUT v Brně, 2016

Vyhledávání trasy v mapě budovy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana inženýra Michala Zachariáše. Další informace mi poskytla firma DataDaemon s.r.o., konkrétně pan Oliver Blšták. Veškeré literární prameny a publikace, ze kterých jsem čerpal, jsem uvedl v seznamu literatury.

.....
Antonín Marko
12. května 2016

Poděkování

Rád bych poděkoval vedoucímu této práce Ing. Michalu Zachariášovi za jeho ochotu vést tento projekt a za jeho trpělivost s mými neustálými dotazy.

Dále bych rád poděkoval firmě DataDaemon s.r.o. za vizi, která dala vzniknout této práci, za poskytnuté technické zázemí a za velmi časté konzultace ohledně samotné implementace.

Obsah

Kapitola 1

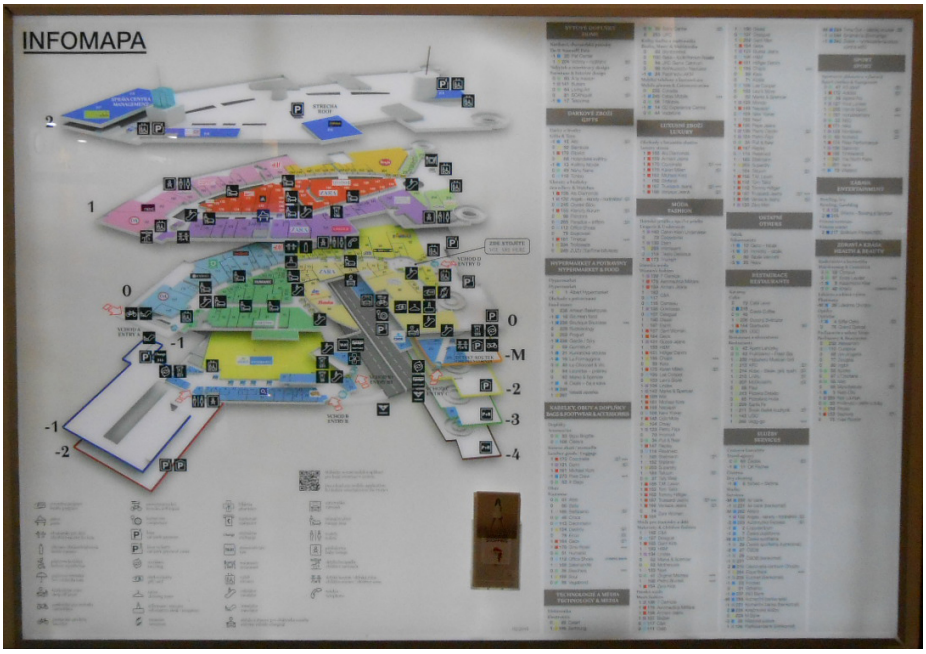
Úvod

V dnešní době plné elektroniky a dotykových zařízení, existuje stále spousta institutů ve velkých budovách, které používají tabule s rozměrnými nalepenými mapami a informačním textem. Člověk, který k této tabuli příjde, se musí nejdříve zorientovat, kde vlastně stojí a následně najít, kam se potřebuje dostat. Případně se musí orientovat pouze podle, někdy velmi zmatených, značek na stěnách nebo na zemi. Zvláště rozlehlé areály s mnoha budovami, jako jsou například nemocnice, vyžadují kvalitní systém zobrazení mapy budovy pro rychlou orientaci.

Velké budovy těmito statickými panely trpí, protože orientace trvá dlouhou dobu. Na obrázcích 1.1 jsou dva typické statické panely, se kterými se můžeme setkat ve vetších obchodních centrech. Plocha panelu je v obou případech rozdělena na část mapy a část legendy. První z map je rozdělena na jednotlivá patra a druhá má vyobrazeno pouze jedno patro. Oba zmíněné panely mají plochu větší než $1m^2$, konkrétně panel 1.1a má plochu o velikosti $2m \times 1,5m$ a panel 1.1b má plochu o velikosti $1,2m \times 1,2m$. S rostoucí velikostí panelů roste i počet zobrazovaných informací, které však postupně zaplňují plochu panelu a ta přestává být přehledná.

Z provedeného pozorování vyplývá, že vyhledávání na těchto panelech probíhá tak, že uživatel nejprve hledá místnost v seznamu v boční části panelu a následně podle přiřazeného čísla hledá konkrétní číslo v mapě. Je to zdlouhavá záležitost. Většinou je mapa zobrazena z jednoho úhlu natočení a není tak zcela jasné, kterým směrem se uživatel dívá a kudy má tedy jít.

Splněným cílem této práce bylo vytvořit plně funkční nástroj pro zobrazení mapy budovy ve 3D a zároveň pro vyhledávání trasy v této mapě. Hlavní myšlenkou projektu je usnadnit prvotní orientaci v budově, ve které uživatel nikdy nebyl a nebo jen potřebuje najít určitou místnost, či osobu.



(a) Chodov Praha

(b) Avion Brno

Obrázek 1.1: Statické panely obchodních center

Kapitola 2

Současná řešení

Během přípravné fáze projektu bylo provedeno pozorování, jak vypadají běžné informační panely v budovách a jak s nimi uživatelé pracují. Navštívené kancelářské budovy měly pouze statický kovový panel s nalepenou 2D mapou, určenou k základní orientaci. Tyto panely měly typicky na okraji legendu s popisky a zbytek plochy panelu zaujímala mapa označená čísly. Zajímavějšími budovami byla obchodní centra, kde jsou informační panely z většiny případů aktivní dotykové plochy. Není to však pravidlem. Vyskytují se i reklamní bannery zobrazující pouze stále se opakující reklamy na aktuální akce v rámci centra.

2.1 Aktivní panely obchodních center

Hardwarové řešení sestává z podstavce a velkého dotykého monitoru s úhlopříčkou okolo 83 cm. Většina monitorů bylo kapacitních s velmi dobrou přesností doteku, ale už ne tak dobrou odezvou[1]. Vyskytují se dva typy naklopení monitoru vůči uživateli, buď svislý monitor a nebo šikmá plocha. Z pohledu uživatele se šikmé řešení používá pohodlněji, svislá plocha je nevýhodná pro osoby nižšího vzrůstu.

Ze zkoumaných 7 panelů byly pouze 3 desktopové aplikace (dále jen aplikace) a zbylé panely zobrazovaly webové stránky. Za účelem získání informací o těchto panelech byla navštívena 4 města, Brno, Praha, Olomouc a Bratislava.

Webové stránky trpěly na možnost kliknutí na externí odkaz a přesměrování na něj, což způsobilo definitivní znemožnění vrátit se zpět na systém centra, následně nutnost zásahu správce a vrácení aplikace do standardního stavu. Na druhou stranu webové stránky vypadají uživatelsky příjemně a pro mnohé uživatele i intuitivně.

Aplikace se ve dvou případech snažila zobrazit víc informací než zvládala a byla několikrát během používání neočekávaně ukončena. Uživatelské rozhraní bylo rozděleno do dvou částí, horní část obsahovala hlavní ovládací prvky aplikace a spodní přehrávala reklamní videa. Pravděpodobně tato kombinace způsobovala pády aplikace při více požadavcích od uživatele, či delším používání. Panel byl uložen svisle i se svisle uloženým monitorem. Při delším sledování používání panelu uživateli bylo zjištěno, že právě lidé nízkého vzrůstu mají problémy s ovládáním horní části panelu, protože na ni nedosáhnou. Tato aplikace byla nainstalovaná na dvou panelech v různých městech, v obou případech měl panel problém s výkonem a při delším používání se aplikace restartovala.

[1] Může být způsobeno monitorem nebo nedostačujícím výkonem hardware.

Třetí aplikace již byla stabilní, zato ale stylizovaná do černé barvy, což na denním světle způsobilo nečitelnost bílých textových popisků a bílých ikon tlačítek.

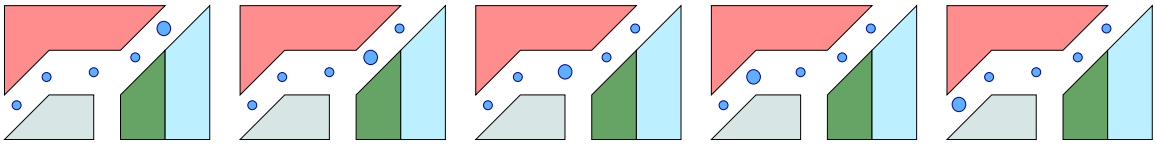
2.2 Zobrazení mapy a vyhledávání tras

U každého z řešení jsem se setkal s různým zobrazováním mapy i realizací vyhledávání, někde vyhledávání vůbec nebylo umožněno. Mapy byly vždy statické, většinou ve 2D, pouze v jednom případě v isometrickém promítání.

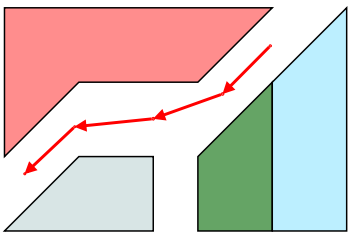
Zobrazované mapy se obecně dají rozdělit do dvou typů, ty které barevně rozlišují jednotlivé místnosti a ty které ne. Pouze jeden panel zobrazoval mapu s barevně odlišenými jednotlivými obchody. V ostatních byl výchozí stav jedné barvy a až po označení se změnila barva, nebo se místnost zasunula do podkladu. Barvy obchodů byly určeny většinou podle typu prodávaného zboží. V jednom případě byla mapa černá na černém pozadí, což se ukázalo jako velmi nepraktické, pokud je za uživatelem zdroj světla, ať už okno nebo zářivka.

Pokud se jednalo o budovu s jedním hlavním patrem a dalšími pouze menšími patry, byla jednotlivá patra zobrazena v jedné ploše a odlišená od hlavního textovými popisky. Častější však je zobrazení ovládacího panelu pro přepínání pater. Tyto panely mají buď vzhled barevných kruhů s čísly pater, nebo konkrétní výpis názvů podlaží.

Vyhledávání trasy bylo umožněno 6 panely různými způsoby. Trasa je vyhledána hned po označení místnosti, nebo je nabídnuto vyhledání trasy v detailu místnosti. Vyhledaná trasa bývá v případě aplikací vždy pohyblivá. Jedna z aplikací zobrazovala trasu kruhovými značkami v mapě, kdy směr trasy byl demonstrován animovaným zvýrazňováním kruhů ve směru trasy (viz obrázek 2.1). Druhý případ animované trasy byl v podobě jasně červených šipek tvaru "V"pohybujících se ve směru trasy. Zobrazení vyhledané trasy bylo provedeno buď celé naráz v podobě lomené čáry, nebo postupným označováním místností, jak by uživatel postupoval. Tento postup však znemožnil s aplikací cokoliv dělat během demonstrace průchodu trasou. Ve webových aplikacích byla trasa většinou zobrazena statickými šipkami, umístěnými v jednotlivých segmentech trasy (viz obrázek 2.2).



Obrázek 2.1: Demonstrace postupného zvýrazňování bodů trasy



Obrázek 2.2: Demostrace zobrazení trasy šipkami po segmentech trasy

2.3 Zhodnocení

Na závěr této kapitoly bych se rád zaměřil na konkrétní oblasti z prostudovaných řešení, které posloužily k ponaučení nebo naopak k získání inspirace do této práce.

Barva pozadí a samotné mapy Černá mapa na černém pozadí zhoršuje čitelnost mapy. Světlé pozadí a barevně odlišené místnosti výrazně zlepšují čitelnost mapy.

Interakce mapy U některých řešení chyběla možnost kliknout na místnost v mapě a zjistit tak, o jakou místnost se jedná. Uživatel tedy mohl tuto informaci zjistit, pouze postupným procházením seznamu nájemníků, případně vyhledáváním podle jména či typu.

Forma mapy Pokud bude mapa ve 2D, na kterou jsou lidé zvyklí z papírových map, měla by mít možnost se otočit ve směru, kterým se uživatel dívá. To platí i pro 3D mapu. Většina uživatelů byla zpočátku zmatena, kterým směrem se vlastně dívá, na mapě byla sice zobrazena jejich pozice, ale už ne kterým směrem je natočen panel.

Animovaná trasa Většinou je trasa statická, složená z šipek nebo pouze z čar. Vyskytují se i řešení, kde trasa uživateli zobrazuje konkrétně, kudy půjde. Může se jednat například o postupné zobrazování místností, kterými musí uživatel projít, než se dostane k cíli.

Ikony a popisky Zobrazení pouze ikony bez textového popisku není ideální řešení, každý člověk chápe význam ikon jinak a dochází tak k častému nenalezení hledané položky. Jedná-li se však o typické ikony (např. toalety, schodiště, výtahy, východy), není potřeba doplňovat textový popis.

Vracení do původního stavu Jedna z realizací měla možnost vrátit se do výchozího stavu aplikace, tedy do hlavního menu. Každá činnost uživatele resetovala počítadlo běžící na horním okraji zobrazení a jakmile počítadlo dorazilo na konec, panel se vrátil na výchozí zobrazení. Této funkce jsem v práci využil pro vytvoření tzv. demo režimu, více v podkapitole 5.4.

Kapitola 3

Použité technologie a struktury v implementaci

Požadavkem na aplikaci bylo, aby běžela na platformě Windows a byla implementována v jazyce C#. S tímto jazykem je úzce spojena platforma .NET Framework[8], která poskytuje bohaté zázemí pro vývoj aplikací na platformu Windows. Knihovna .NET, jazyk C# i další jazyky, jsou kvalitně zdokumentovány na portálu MSDN, ze kterého jsem také čerpal.

Jazyk C# je objektově orientovaný jazyk vyvinutý firmou Microsoft zároveň s platformou .NET Framework kolem roku 2000. Vychází z jazyků C++ a Java, z jazyka C přebírá syntaxi. Tento jazyk je určen k tvorbě desktopových a mobilních aplikací, databázových programů, webových stránek, atd.

Podstatnou částí vývoje v C# byla implementace návrhového vzoru Model View ViewModel (zkráceně MVVM)[4]. Tento návrhový vzor byl vybrán z důvodu oddělení datové části od logické části, čímž bylo dosaženo snadnější modifikovatelnosti jednotlivých částí. Na návrhový vzor MVVM je navázán grafický podsystém Windows Presentation Foundation[9], který byl přímo pro MVVM připraven.

3.1 Windows Presentation Foundation

Jedná se o grafický podsystém pro vykreslování uživatelského prostředí pro Windows obsažený v .NET Frameworku od verze 3.0, který primárně využívá XAML[5] pro definici rozložení komponent v prostředí. Přímo nahrazuje starší podsystém WinForms, který byl založen na GDI+. WPF využívá DirectX pro vykreslování a je tedy umožněna hardwarová akcelerace na grafické kartě.

WPF bylo zvoleno místo zastaralého WinForms, protože nativně podporuje MVVM, umí vykreslovat 3D objekty do komponenty k tomu určené a všeobecně se jedná o moderní technologii.

Vykreslování 3D modelů bylo rozhodujícím parametrem při výběru prostředí. V počátcích bylo uvažováno nad použitím Unity3D, které je velmi silným nástrojem pro tvorbu 3D aplikací, především her. Ovládání aplikace je řešeno přes skripty v mnoha jazycích, mezi které patří i C#. Vzhledem k požadavku, že má být aplikace implementována v C# a integrovatelná do již stávající aplikace, bylo použití Unity3D zavrženo.

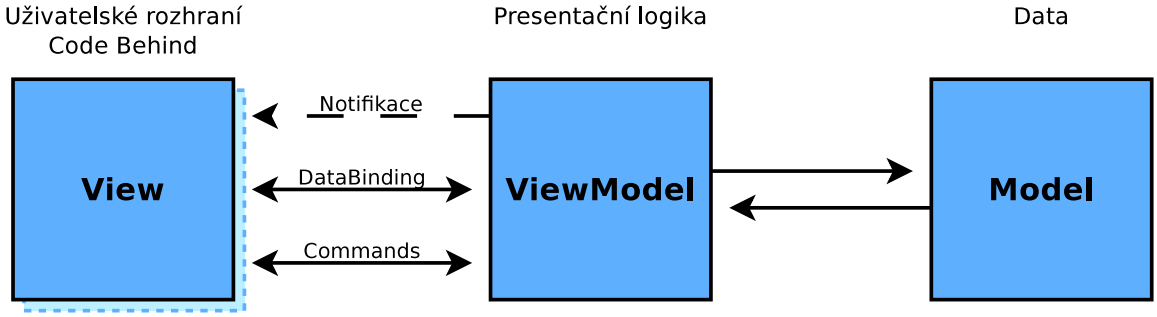
Literaturou pro práci s WPF byla hlavně kniha WPF Recipes in C# 2008[7] obsahující mnoho příkladů, mezi které patří i základy práce s 3D modely.

3.2 Návrhový vzor Model View ViewModel

Principem tohoto návrhového vzoru je oddělení uživatelského rozhraní, aktivního kódu a datových struktur. Poskytuje tím snadnější správu jednotlivých částí.

MVVM dědí z předchozího Model-View-Controler návrhového vzoru, který je zaměřený na implementaci uživatelských rozhraní pro desktopové aplikace.

Specifické pro MVVM jsou 3 hlavní celky a to Model, View a ViewModel (někdy též nahrazováno pojmem Binder). Každý z těchto celků mezi sebou určitým způsobem komunikují. Na diagramu 3.1 jsou znázorněny celky a způsoby komunikace mezi nimi.



Obrázek 3.1: Návrhový vzor Model View ViewModel

Z diagramu vyplývá, že View a ViewModel jsou nejvíce propojeny z pohledu komunikace. ViewModel s Modely komunikuje na úrovni získávání datových struktur.

Model popisuje data a struktury, které se vyskytují v aplikaci. V případě této aplikace téměř kopírují tabulky databáze a reprezentují jejich obsah. Avšak model nesmí o aplikaci nic vědět, slouží pouze jako datová struktura pro uchování dat.

View reprezentují veškerá uživatelská rozhraní aplikace, ať už se jedná o okna, formuláře nebo položky seznamů se specifickým rozložením. Tato View jsou definována ve WPF a k View Modelu vázána pomocí Data Bindingu.

ViewModel obsahuje hlavní logiku aplikace, tvoří spojovací vrstvu mezi View a Modely. Pomocí bindingu váže nastavení komponent ve View na vlastnosti (Property) uvnitř ViewModelu. Dále pomocí příkazů (Command) volaných z View umožňuje zachytávat kliknutí na tlačítko apod. Hlavní podmínkou pro chod vzájemné interakce View a ViewModelu je implementace rozhraní INotifyPropertyChanged.

S implementací MVVM však přichází i mnoho omezení, například zamezení využívání Code-Behind u komponent, protože to porušuje podmínku, že aktivní kód je obsažen pouze ve ViewModelu. V některých případech je ale použití Code-Behind jediný způsob, jak nějakou činnost vykonat.

3.3 Datové struktury aplikace

Aplikace je rozdělena do několika samostatných projektů, které vytvářejí mezi sebou komunikující soustavu knihoven. Jednotlivé od sebe oddělitelné celky projektů jsou uspořá-

dány do složek podle názvů a tedy do vlastních jmenných prostorů. Jednotlivé projekty jsou nazvány složením jména práce, tedy BuildingViewer, a názvem konkrétního projektu.

BuildingViewer.Frontend

Hlavním projektem je BuildingViewer.Frontend, který obsahuje veškeré UserControl komponenty utvářející uživatelské rozhraní. Tento projekt implementuje View část MVVM návrhu a jedná se o projekt s označením *StartUp project*, což znamená, že se jedná o spustitelný projekt, jehož výstupem je EXE soubor. Tento projekt jako jediný má reference na ViewModely i Modely.

BuildingViewer.Models

Druhým projektem spolupracujícím na MVVM je BuildingViewer.Models, obsahující modely odpovídající datovému návrhu. Modely slouží pouze jako datové šablony, projekt tedy nevlastní žádné reference na ostatní MVVM projekty.

BuildingViewer.ViewModels

Poslední projekt, který přispívá do MVVM návrhu je projekt BuildingViewer.ViewModels, který obsahuje logiku aplikace. Referenci má na Modely, ze kterých čerpá struktury, ale na View vazbu mít nesmí.

Building.Viewer.Database

Prvním projektem nespolupracujícím na MVVM návrhu je projekt BuildingViewer.Database, který obsahuje třídy pro konektivitu s databází, kontroly aktuálnosti dat, apod.

BuildingViewer.Statistics

Jediný projekt ukládající data do databáze je projekt BuildingViewer.Statistics, určený k zapisování statistických informací z používání aplikace.

BuildingViewer.Logging

Tento projekt obstarává ukládání informací o chodu aplikace do systémového EventLogu nebo do textového souboru, obě možnosti je možné zapínat a vypínat z config souboru.

BuildingViewer.Config

Tímto se dostáváme k projektu BuildingViewer.Config, který obsluhuje definice jednotlivých sekcí a elementů potřebných k uchování nastavení konkrétní instance aplikace. Obsahem projektu je rozšíření standardního konfiguračního souboru o vlastní sekce s vlastními elementy pro jejich konkrétní využití v různých částech aplikace.

BuildingViewer.Pathfinding

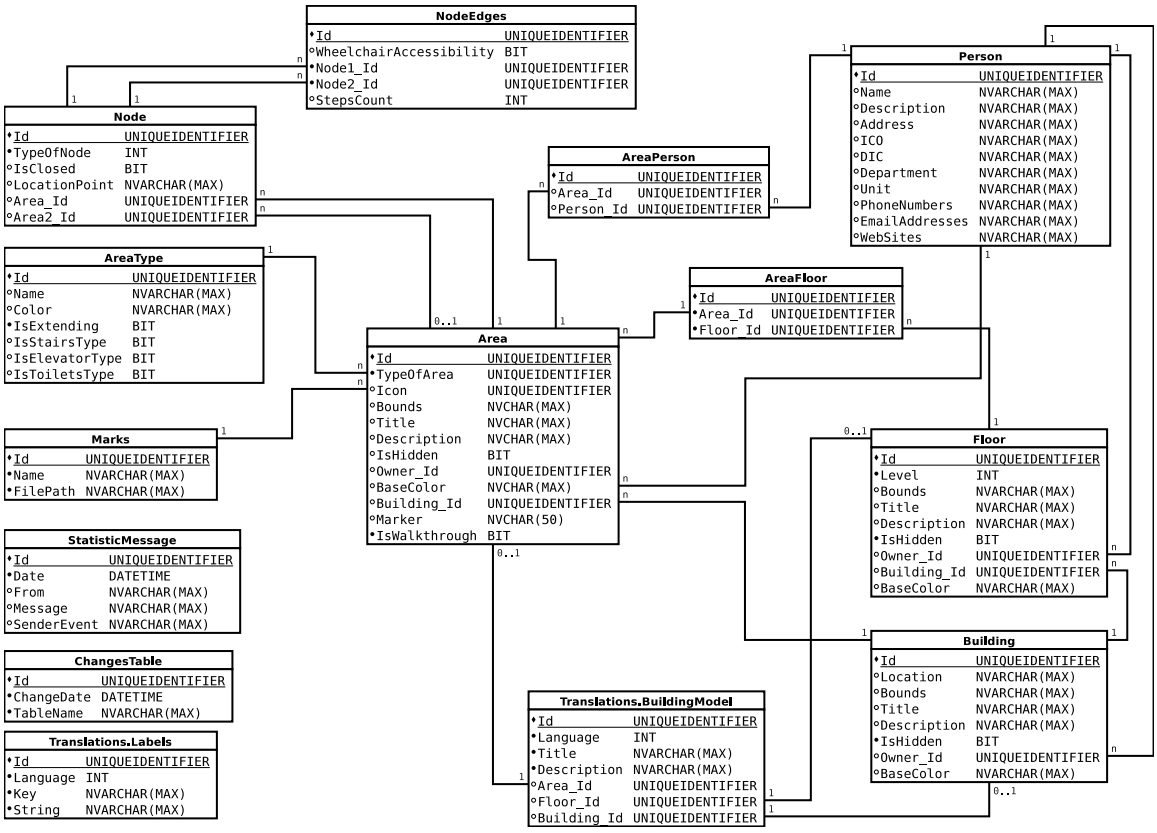
Posledním projektem této aplikace je BuildingViewer.Pathfinding, který obsahuje implementaci vyhledávacích algoritmů a třídy Pathfinder, která toto vyhledávání zprostředkovává.

Kapitola 4

Datový návrh – databázová část

První část implementace projektu bylo sestavení správného datového návrhu a následně vytvoření databáze. Celá aplikace pracuje s daty primárně z databáze.

Každá tabulka zastupuje určitou část reálného systému, uvažováno bylo nad budovou, která obsahuje patra a na těchto patrech se vyskytují místnosti. V těchto místnostech se vyskytují osoby. Určitá osoba může každou z předchozích částí vlastnit. Místnosti obsahují množiny bodů, které mají určitý účel. Mezi těmito body jsou vytvořeny spoje, po kterých probíhá vyhledávání trasy.



Obrázek 4.1: Entity relationship diagram databáze

Do budoucna je počítáno s rozšířením datového návrhu pro specifická řešení, současný stav ale pokrývá základní funkčnost aplikace.

4.1 Lokalizace aplikace

Standardním způsobem implementace jazykových mutací aplikací ve WPF je používání Resources a knihoven k tomu určených. Tento projekt však předpokládá více souběžně běžících aplikací, kde by v případě úpravy lokalizace bylo nutné do všech aplikací tento překlad nahrát a aplikaci restartovat.

Řešením bylo centralizovat lokalizace do databáze a všechny změny načítat z jednoho místa. Proto byly v databázi vytvořeny dvě tabulky Translations.Labels a Translations.BuildingModel. První z nich slouží k překladům různých popisků uživatelského rozhraní a druhá k překladům textů objektů budovy.

4.2 DataReader – třída pro čtení databáze

Na počátku vývoje bylo uvažováno nad využitím EntityFrameworku [12] jako mezivrstvy pro čtení a zápis do databáze. Bohužel se ani opakovaně nepodařilo tento framework zprovoznit, proto byla zvolena možnost navrhnout a vytvořit třídu, která bude data z databáze načítat a plnit do C# objektů.

Jelikož aplikace pracuje pouze jako zobrazovací člen, z databáze pouze čte. Zápisy jsou prováděny pouze v případě statistických záznamů, a to v samostatném projektu.

Potřebnou mezivrstvu tvoří třída DataReader, která komunikuje s databází a kompletuje data z jednotlivých tabulek do C# objektů. Postup čtení je rozdělen do 3 částí: naplnění třídy DataSet, naplnění tříd modelů z dat v DataSetu a provázání vzniklých objektů podle cizích klíčů. Po dokončení všech částí DataReader obsahuje veškerá data z databáze převedená do C# objektů včetně všech vazeb.

DataReader implementuje návrhový vzor singleton (jedináček), čímž je zajištěno, že data, která obsahuje, jsou aktuální a jediná v aplikaci. Implementace je řešena statickou instancí třídy DataReader, která při inicializaci načte veškerá data z databáze. Inicializace je spuštěna po startu aplikace a každý view model si referenci na DataReader uchovává ve vlastnosti Provider, kterou obsahuje bázová třída view modelů BaseViewModel.

Obsahem DataReaderu je i kontrolní sekce, která v určitých intervalech, daných hodnotou v konfiguračním souboru, kontroluje změnovou tabulku databáze, kam se pomocí databázových triggerů ukládají záznamy o změnách (INSERT, UPDATE a DELETE) nad kontrolovanými tabulkami. S každou proběhlou změnou se uloží její čas do DataReaderu a ten si při dalším čase kontroly příkazem SELECT vyžádá záznamy s novějším časem než je čas poslední kontroly. Následně se provede načtení všech dat se změnami do DataSetu a provázání s ostatními tabulkami. Tímto principem je zajištěna aktuálnost dat zobrazovaných všemi běžícími aplikacemi.

4.3 Databázové tabulky

Tabulky obsažené v datovém návrhu zastupují entity z reálných systémů. V této kapitole rozeberu jednotlivé tabulky, jejich obsah a důvody jejich vzájemných vazeb.

Část tabulek vychází ze stejné šablony, která popisuje část budovy, tato šablona byla pojmenována BuildingModel. Specifikem jsou položky Bounds, Title, Description, Owner_Id, IsHidden a BaseColor. Tabulky, které tuto šablonu dědí a rozšiřují, jsou Area, Floor a Building. Tyto tři tabulky tvoří hlavní datové jádro pro uchovávání informací o budově.

Položka Bounds vymezuje půdorys objektu, data jsou uchována v podobě pole bodů. Souřadnice X a Y bodů jsou odděleny čárkou a jednotlivé body pak středníkem. Title a Description jsou textové údaje o místnosti, název a popis, které slouží jako výchozí hodnoty lokalizace. Přepínač IsHidden určuje, zda-li má být konkrétní objekt zobrazen uživateli, např. sklad nebo místnosti, kam nemá přístup. BaseColor určuje barvu místnosti, pokud je zadaný má vyšší prioritu než výchozí barva typu místnosti. Poslední společnou položkou je Owner_Id, jedná se o odkaz na tabulku Persons, který vyjadřuje majitele, či nájemníka konkrétní entity.

Area – místnost

Jak už z názvu vyplývá, jedná se o definici místnosti. BuildingModel rozšiřuje o položky TypeOfArea, Icon, Building_Id, Marker a IsWalkthrough. Položka TypeOfArea odkazuje na tabulku AreaType, Icon odkazuje na tabulku Marks a určuje logo místnosti, Building_Id odkazuje na tabulku Building, přepínač IsWalkthrough určuje, jestli je místnost průchozí. Poslední položkou je Marker, oproti Icon se jedná o text odkazující se do konfiguračního souboru aplikace, kde jsou určeny obrázkové zdroje pro rotující značky.

AreaType – typ místnosti

Tabulka AreaType slouží k sjednocení místností stejného typu do množin se stejnými vlastnostmi. Stejné vlastnosti jsou především výchozí barva místnosti v položce Color a nebo možnost vysunutí místnosti po označení v položce IsExtending. Následující tři přepínače určují speciální typ místnosti, jsou to toalety (IsToiletsType), schodiště (IsStairsType) a výtahy (IsElevatorType).

Při vytváření modelu místnosti je barva materiálu primárně brána z položky Color této tabulky. Dále pak položka IsExtending rozděluje modely místností do dvou kategorií, které pak zásadně určují chování modelu (viz 5.2.2).

Marks – značky pro místnosti, ikony, loga

Místnost může mít konkrétní ikonu, logo nebo piktogram, které zobrazuje v detailu. Cestu k jednotlivým obrázkům obsahuje položka FilePath a název značky pak položka Name.

Obrázek definovaný touto položkou se zobrazuje v detailu místnosti jako logo. Do budoucna je plánováno rozšíření tabulky o definici plochy a natočení vůči určitému bodu v půdorysu místnosti, aby bylo možné toto logo zobrazit přímo v mapě.

Node – body mapy

Dveře nebo například východy mají v databázi zastoupení v podobě tabulky Node, ta obsahuje položky jako LocationPoint, což je umístění bodu v dvou dimenzionální soustavě souřadnic, TypeOfNode reprezentující typ bodu, přepínač IsClosed v kombinaci s typem dveří určuje, jestli je možné skrz dveře vyhledávat trasu. Poslední dvě položky, Area_Id a Area2_Id mají dvojí využití, jestliže se jedná o dveře, musí být určeny oba klíče, protože dveře spojují právě dvě místnosti, v ostatních typech bodů je určen pouze klíč Area_Id, což je místnost, které bod náleží.

NodeEdges – spojnice bodů

Všechny body budovy tvoří uzly grafu pro vyhledávání, mezi uzly bylo nutné vytvořit hrany, tyto hrany jsou definovány v tabulce NodeEdges. Hrana je vytvořena mezi právě dvěma body a obsahuje další informace, jako například přepínač, jestli se jedná o bezbarierovou hranu, a nebo kolik má schodů na celou délku hrany. Více v kapitole 6 o Vyhledávání trasy.

Floor – patro

Tabulka Floor popisuje patro budovy a rozšiřuje BuildingModel pouze o položku Level a Building_Id. Level představuje číslo podlaží, záporné hodnoty odpovídají podzemním patrům, kladné nadzemním a nula přízemí. Building_Id odkazuje na budovu, do které toto patro patří.

AreaFloor – vazba místnosti na patro

Každá místnost se nachází na určitém patře, ale existují místnosti, které prochází skrze patra, například schodiště nebo výtahy.

Při vytváření místností schodišť a výtahů bylo nutné je svázat s každým patrem a následně vytvořit jednotlivé vchodové dveře. V případě výtahů to představovalo vytvoření řady dveří nad sebou, protože výtah se pohybuje pouze po jedné ose. Kdežto schodiště (příp. eskalátor) má vstupy dva a to každý v jiném bodě na mapě. Mezi těmito dvěma body na patrech bylo nutné vytvořit spojnici NodeEdges a podle typu doplnit parametry. Jednalo-li se o výtah, pak je nutné přepínač WheelchairAccessibility nastavit. Schodiště bezbarierové není a navíc je nutné zadat počet schodů. Vycházíme-li z předpokladu, že schodiště dodržují normy, pak podle počtu schodů můžeme určit délku trasy po schodišti, viz 6.1.2

Building – budova

Poslední z tabulek, které rozšiřují šablonu BuildingModel, zde pouze o položku Location, která zastupuje popis místa (adresu, GPS souřadnice, apod.), kde se budova nachází.

Definice budovy jako takové v aplikaci vpodstatě nemá význam, protože předpokládáme, že vyhledávání trasy bude probíhat v rámci jedné budovy. Do budoucna se však počítá s areálem budov, kdy vyhledávání bude umožněno i mezi budovami apod.

Person – osoba, fyzická nebo právnická

Podstatnou tabulkou z pohledu vyhledávání je tabulka Person, zastupující jak fyzické osoby, tak i právnické. Je možné tedy jednou tabulkou definovat osobu, která se může nacházet někde v budově a nebo firmu, která budovu vlastní apod.

Jako základní položky tabulky jsou Name, Description a Address obsahující obecný popis osoby, či firmy. Položky PhoneNumbers, EmailAddresses a WebSites obsahují serializované seznamy hodnot oddělených středníky, které aplikace po přečtení převádí do podoby seznamů.

Další dvě dvojice parametrů záznamu určují detailnější informace, buď o osobě, nebo o firmě. Pokud záznam popisuje osobu, mohou být vyplněny položky Unit a Department popisující, kam osoba v rámci hierarchie firmy patří. Pokud záznam popisuje firmu, k dispozici jsou pole pro IČO a DIČ.

AreaPerson – vazba místnosti a osob

Předpokladem u vztahu místnosti a osob bylo, že se v místnosti může vyskytovat víc jak jedna osoba, ale jedna osoba může být registrovaná ve více místnostech. Například může mít osoba kancelář v přízemí, ale laboratoř v patře, kdy kancelář v přízemí sdílí s jinou osobou.

Translations.Labels – překlady popisků aplikace

První z lokalizačních tabulek s doplňkovým názvem Labels obsahuje překlady popisků tlačítek, popisků textových polí apod. Klíčem k určení, pro který popisek je překlad určen, je položka Key, jejímž obsahem je název elementu definovaný v XAML. Celočíselné označení jazyka překladu je uchováno v položce Language a samotný text překladu v položce String.

Translations.BuildingModel – překlady částí budovy

Druhá lokalizační tabulka obsahuje překlady objektů dědících šablonu BuildingModel, což jsou tabulky Area, Floor a Building. Jelikož je u těchto objektů potřeba překládat pouze položky Title a Description, tato tabulka obsahuje příslušná pole taktéž. Pro kterou tabulku je příslušný překlad určen specifikuje jedna z položek Area_Id, Floor_Id nebo Building_Id. U této tabulky je snadné do databáze zanést chybu, protože je možné vložit současně 3 cizí klíče. Tento případ musí být ošetřen v administrátorské aplikaci, která bude vytvářet INSERT skripty. Případně může být chybový stav kontrolován validační procedurou přímo v rámci databáze.

Položka Language má stejný význam jako u překladů popisků, konkrétní číselné hodnoty jsou uloženy v konfiguračních souborech, kde je zadána i cesta k obrázku vlajky apod.

ChangesTable – změnová tabulka

Na určité tabulky jsou navázány databázové triggery, které vytvářejí záznamy v této tabulce po operacích INSERT, UPDATE a DELETE. Jde o systém uchovávání stálé aktuálnosti aplikací navázaných na databázi. Pokud administrátor provede změnu v datech, změnová tabulka tyto změny zaznamená a běžící aplikace všechny změny načtou z databáze.

Datum každé změny určuje, jestli se jedná o novou změnu, kterou je potřeba načíst, nebo o změnu, která již načtena byla. Aplikace si v DataReaderu uchovává čas poslední načtené změny, pokud ve změnové tabulce nalezne záznamy s novějším časem, začne načítat data.

StatisticMessage – zaznamenávání statistických údajů

Předpokladem do budoucnosti je zobrazování statistik používání panelu, resp. počty vyhledávání určitých místností či osob, počet kliknutí na určité místnosti, apod. Dále pak přizpůsobení demo režimu na statistické údaje, např. častější zobrazování často hledaných místností, apod.

Jedná se o jedinou tabulku, která je plněna z aplikace. K vytváření záznamů slouží statická třída, která je volaná z místa, kde událost vznikla.

Položka Date určuje datum a čas, kdy k události došlo. Položka From reprezentuje konkrétní aplikaci a pole Message obsahuje textový, ale i strukturovaný popis události. SenderEvent reprezentuje konkrétní spouštěcí údálost, která vyvolala záznam.

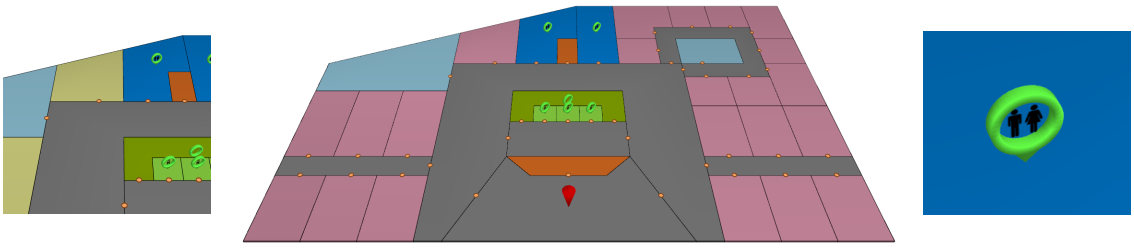
Kapitola 5

Prezentační část

Zobrazení mapy ve 3D, oproti standardním 2D pohledům shora, je z mnoha ohledů názornější a lepší pro orientaci v prostoru. Pokud uživateli zobrazíme mapu natočenou v jeho úhlu pohledu a nakloněnou s jasným perspektivním promítáním, měl by být lépe a rychleji orientovaný ve vyobrazené mapě.

K lepší orientaci v jednotlivých modelech (místnosti, chodby, apod.) mapy slouží i jejich barevná odlišnost. Aplikace umožňuje modely stejného typu obarvit barvou definovanou v tabulce typů, ale také nastavit barvu podle bázové barvy místnosti, pokud je nastavená. Je tedy možné modely jednoho typu dále odlišit definovanou barvou, např. místnost patřící určité firmě, která má specifikované firemní barvy, barvou místnosti tedy bude jedna z nich.

Speciální místnosti, jako například toalety, schodiště nebo výtahy, jsou v mapě zvýrazněny rotující značkou se symbolem odpovídajícího typu. Není tedy nutné procházet mapu, aby uživatel nalezl toalety, ale hned na výchozím zobrazení uvidí symbol toalet.



Obrázek 5.1: Různé barvy místností podle typu, vzhled kompletní mapy a rotující značky

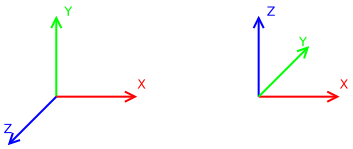
Zásadním mezníkem byla grafická část aplikace, konkrétně vykreslování 3D scény s mapou budovy, jednotlivých místností a jejich animací. Do modelů můžeme počítat jednotlivé značky mapy, jako například značky dveří, bod *zde stojíte*, a nebo vyhledanou trasu.

S využitím WPF je usnadněno vytváření, upravování i zobrazování 3D grafiky, buď z jazyka XAML, nebo rovnou z kódu. Vzhledem k tomu, že mapa je proměnlivá, nelze využít zadávání modelů pomocí XAML, ale musí být modely generovány procedurálně.

5.1 Úvod do 3D pomocí WPF

WPF poskytuje mnoho tříd pro práci s 3D grafikou. Jako základní prvky pro práci jsou body, vektory a matice, všechny doplněny o třetí dimenzi oproti třídám pro 2D grafiku.

Orientace jednotlivých os v prostoru je pro každý grafický software odlišný. WPF je založeno na pravidle pravé ruky[1] a využívá předpoklad, že renderovací plocha je typicky obrazovka, souřadnice X a Y jsou pak na svislé ploše a osa Z je hloubkou pohledu. K porovnání poslouží 3D modelovací a renderovací software 3D studio MAX[10], který má osy uspořádané tak, že podstava je tvořena osami X a Y a výška modelu osou Z, viz obrázek 5.2. Taktéž využívá pravidlo pravé ruky, jen jsou jednotlivé osy otočeny kolem osy X, aby osa Z směřovala vzhůru.



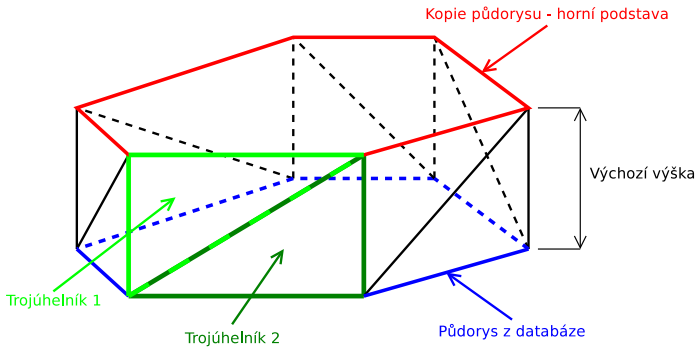
Obrázek 5.2: Uspořádání souřadnicových os používaných ve WPF a 3DsMAX

Komponentou pro vykreslování 3D scény ve WPF je Viewport3D. Obsahuje vlastnost Children typu Visual3DCollection, která obsahuje objekty implementující abstraktní třídu Visual3D. Tyto objekty mohou být samotné 3D modely nebo zdroje světla. Vlastnost Children však nelze navázat na vlastnost[2] v příslušném ViewModelu, pokud je potřeba dynamicky měnit obsah komponenty Viewport3D, pak je nutné porušit MVVM návrh a změny provést v CodeBehind.

Kamera může být buď ortografická, nebo perspektivní, a je přidávána do vlastnosti Camera komponenty Viewport3D. Kameru je možné přidávat jak z XAML, tak z kódu ViewModelu. Pohyby kamery jsou řízeny dotykem, či myší. Pohyb po 2 osách je nutné převést do 3D prostoru a pohnout kameru určitým směrem. Výchozí natočení kamery je specifikováno v konfiguračním souboru u definice bodu *zde stojíte* pro konkrétní instanci aplikace. Tento úhel se během používání aplikace nemění.

5.2 3D modely

Definice 3D modelů je uložena v databázi jako půdorys místnosti.



Obrázek 5.3: Demonstrace konverze 2D půdorysu do 3D modelu

Na obrázku 5.3 je popsán postup, jakým jsou modely místností vytvářeny. Modrý polygon je půdorys místnosti z databáze, zadávaný formou seznamu bodů. Červený polygon

[1] Jednotlivé prsty indikují pozitivní směr os, ukazováček osu X, prostředníček osu Y a palec osu Z

[2] Vlastnost Children má pouze getter, do kolekce Visual3DCollection se dají objekty vkládat pouze funkcí Add() nebo Insert().

je horní podstava vysunutá o hodnotu odpovídající výchozí výšce modelu v neoznačeném stavu. A oba zelené trojúhelníky tvoří jednu stěnu modelu. Nad červeným polygonem probíhá triangulace pomocí EarClipping algoritmu, popsaným níže.

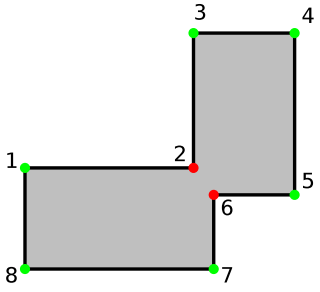
Algoritmy pro tvorbu základních 3D modelů pro modelování trasy nebo různých značek ve scéně jsou uloženy jako statické funkce ve třídě Objects3D. Prototypování modelů přímo v XAML by bylo téměř nemožné, protože se jedná o modely s větším počtem vrcholů a výsledných trojúhelníku, které je jednodušší počítat procedurálně než zadávat do XAML.

5.2.1 EarClipping algoritmus

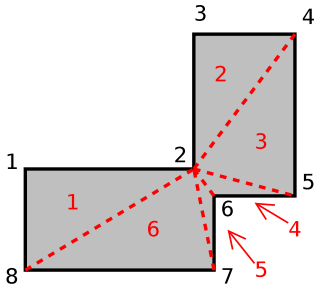
Principem algoritmu je odstraňování vrcholů polygonu, které se svými sousedními vrcholy tvoří trojúhelník uvnitř polygonu.

Vstupní polygon je zadán v podobě pole bodů v 2D prostoru. Výstupem je množina trojúhelníků tvořících vstupní polygon.

Algoritmus rozlišuje 2 základní typy vrcholů polygonu, konvexní a reflexní. Konvexními vrcholy jsou všechny vrcholy, které s předchozím a následujícím vrcholem svírají vnitřní úhel menší jak 180°. Reflexní vrcholy jsou naopak ty, které s předchozím a následujícím vrcholem svírají vnitřní úhel větší jak 180°, viz obrázek 5.4. Speciálním typem konvexního vrcholu je tzv. *ear*, jedná se o vrchol, který se svými sousedními vrcholy vytváří trojúhelník, který leží uvnitř polygonu a zároveň neobsahuje žádný z jiných vrcholů polygonu.



Obrázek 5.4: Konvexní (zelené) a reflexní (červené) vrcholy



Obrázek 5.5: Vznikající trojúhelníky

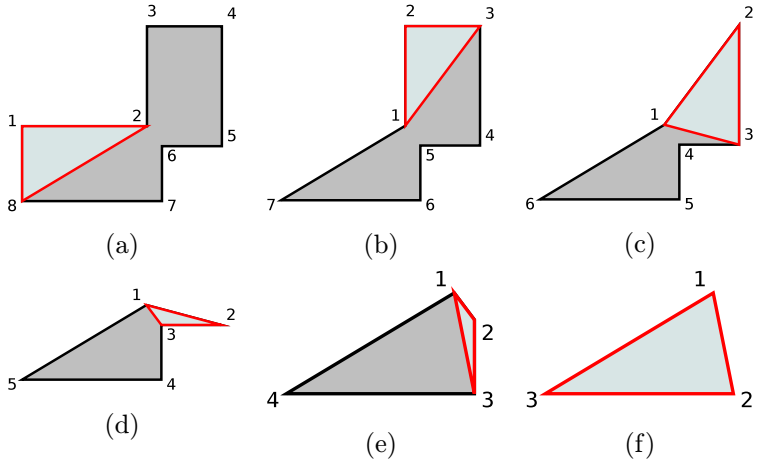
Algoritmus sestává z hlavních 3 bodů:

1. Pokud je počet bodů polygonu menší jak 3, konec
2. Určit Konvexní, Reflexní a *Ear* vrcholy
3. Vytvořit trojúhelník z prvního *Ear* vrcholu, trojúhelník uložit a vrchol odebrat, opakovat bod 1

Tento postup je popsaný v článku [3], v projektu je však mírně upravený. Úprava spočívá v tom, že se neprovádí průchod pro zjištění, které konvexní vrcholy jsou *ear* a až následně další průchod na odstranění těchto vrcholů. Provede se pouze jeden průchod, během kterého se hned po detekci *ear* vrcholu vrchol odstraní. Je tím ušetřen jeden průchod celým polem.

Jednotlivé kroky algoritmu jsou demonstrovány níže na obrázku 5.6.

Tímto algoritmem jsou převáděny všechny modely z půdorysu v databázi do 3D modelů v aplikaci. Je spouštěn vždy, když se načítá patro budovy, nad každou místností zvlášť.



Obrázek 5.6: Jednotlivé kroky algoritmu demonstrované na jednoduchém polygonu

5.2.2 Klasifikace 3D modelů budovy

Budova je rozdělena na podlaží a ta jsou dále dělena na místnosti. Budova jako celek model nemá, podlaží v podstatě také ne. Modely mají až místnosti na konkrétním podlaží. Tyto modely je možné rozdělit do dvou typů, expandující a neexpandující. Rozdělení se vztahuje k změně výšky modelu jako reakce na označení modelu. Označení se projeví na modelu i tak, že změní svou barvu. Výchozím stavem jsou zašedlé nevýrazné barvy, po označení se barva nastaví na barvu nastavenou v databázi. Změna barvy, konkrétně nevýraznost barvy, je provedena získáním 80% z původní barvy a jejích jednotlivých RGB složek. Výsledná barva je pak mírně tmavší a není tolik výrazná.

Expandující modely Mezi tyto modely se řadí všechny místnosti, ke kterým je možné vyhledat trasu. Reakcí na označení je jejich vysunutí z mapy podlaží. Vysunutí je provedeno jednoduchou animací Y osy, resp. aplikování tzv. scale trasformace na Y osu modelu. Pokud je objekt označen, je spuštěna rostoucí animace, pokud objekt označení ztratí, animace je spuštěna opačným směrem.

Neexpandující modely Typicky chodby a haly jsou neexpandující modely, k těmto místnostem není možné vyhledávat trasu a na označení reagují pouze změnou barvy.

5.2.3 Interakce modelů

Označování modelů je primárně prováděno uživatelem a to kliknutím na příslušný model. Toto kliknutí však není možné nijak navázat DataBindingem, protože Viewport3D neobsahuje žádnou vlastnost SelectedModel apod. Bohužel zde bylo nutné znovu porušit MVVM a odchytávat kliknutí pomocí eventů. Třída ModelVisual3D byla nahrazena třídou ModelUIElement3D, která přímo podporuje odchytávání vstupních eventů. Každý model tedy implementuje MouseDown, MouseUp a MouseMove. V těchto procedůrách je řešeno označení modelu, ale i posun kamery. Změna označení modelu se provede pouze, pokud se body z MouseDown a MouseUp neliší víc, jak o určitou hodnotu. V opačném případě je to vyhodnoceno jako pohyb kamery.

Tím, že se označení děje uvnitř modelu, je nutné aplikaci sdělit, který model a v jakém stavu se nachází. Zde opět nešlo využít DataBindingu, protože se jedná o model uvnitř

Visual3DCollection. Každá změna označení modelu se do aplikace přenese pomocí eventů, které jsou vyvolány po konkrétních operacích.

5.2.4 Knihovna 3D objektů – Objects3D

WPF jako takové neposkytuje žádné funkce pro vytvoření primitivních 3D objektů, proto byla implementována sada funkcí ve statické třídě Objects3D. Obsahem jsou metody pro vytvoření hranolů, koulí, šipek, kuželů apod. S vytvářením této knihovy jsem našel inspiraci na webové stránce `csharphelper.com`[11], kde je prakticky řešeno mnoho příkladů z oblasti 3D grafiky ve WPF. Některé z těchto algoritmů popíšu zde.

Většina modelů popisovaných níže je osově souměrných, jedná se buď o modely s kruhovým tvarem, nebo o modely vycházející z jednoho bodu. Kruhové modely jsou například model indikující dveře a nebo model značek. Ostatní modely vycházejí z bodu, většinou se jedná o střed nebo počátek.

Kužel Model kuželu byl řešen jako jeden z prvních modelů vůbec. Sestává z podstavy a pláště. Výhodou je, že jej můžeme celý složit z podobných trojúhelníků. Podstava jsou trojúhelníky jdoucí od středu kužele k obvodu podstavy a plášť jsou trojúhelníky od obvodu podstavy k vrcholu kužele. Kužel je možné parametrizovat, co se týče jeho výšky a průměru podstavy. Viz obrázek 5.7a.

Úsečka V 3D prostoru je možné úsečku znázornit více způsoby, v této práci byl zvolen hranol jako reprezentační model pro úsečku, jelikož je složen z malého počtu trojúhelníků. Úsečka se modeluje mezi dvěma body v prostoru, tudíž je nutné určit potřebné úhly pro natočení modelu a jeho bodů. Každá z linek se skládá z 8 bodů, mezi kterými jsou nakonec vytvořeny trojúhelníky. Viz obrázek 5.7b.

Koule Nejsložitějším objektem v této třídě je koule. Pro výpočet jednotlivých bodů na povrchu koule byly použity následující rovnice:

$$x = r * sin\ p * cos\ t \tag{5.1}$$

$$y = r * sin\ p * sin\ t \tag{5.2}$$

$$z = r * cos\ p \tag{5.3}$$

Proměnná r je poloměr koule a je definovaná jako parametr funkce. Parametry p a t jsou úhly popisující aproximaci na jednotlivých osách. A proměnné x, y a z jsou souřadnice jednoho bodu na povrchu koule v konkrétním stavu.

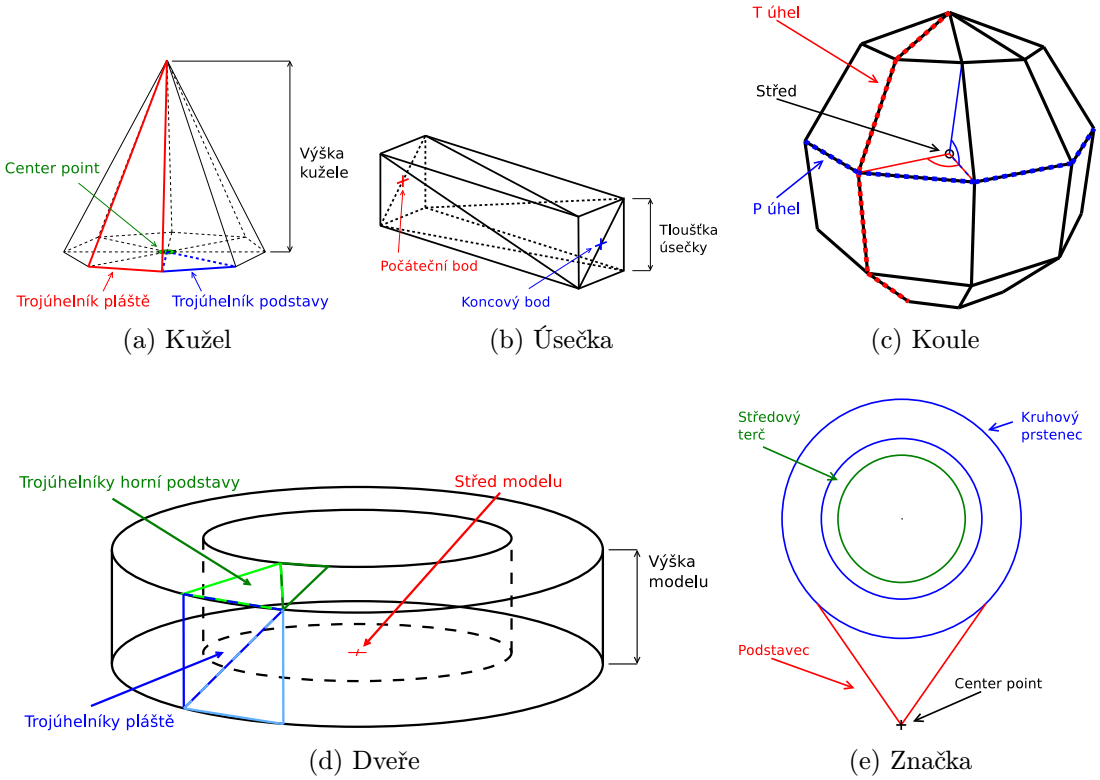
Jedná se o aplikování rovnic sférické soustavy souřadnic na povrch koule k získání konkrétních souřadnic na stejných místech relativně k počátku. Viz obrázek 5.7c.

Šipky Složenými modely jsou například šipky, skládají se z kuželu a válce. Model je složen z 3 kružnic a dvou bodů určujících počátek šipky a její vrchol. Každá šipka se tedy skládá z kruhové podstavy, válcové části, prstence před samotnou šipkou a kuželovitého vrcholu.

Dveře Značka pro dveře sestává z kruhového prstence a kruhové podstavy. Prstenec je složen ze 4 kružnic, mezi kterými jsou utvořeny plochy z trojúhelníků. Viz obrázek 5.7d.

Značky Do mapy jsou zabudovány otáčející se značky s určitým informačním obrázkem. Tyto značky sestávají z kruhového prstence, podstavce a středového kruhového terče. Podstavec je složen z trojúhelníků navazujících na spodní část prstence a mají společný bod v centrálním bodě předaným parametrem. Viz obrázek 5.7e.

Stíny Nejedná se o konkrétní model, ale spíše o obalovací funkci, která vezme model z parametru a přidá mu kruhový stín (blob shadow). Stín je natažen jako textura na čtvercovou plochu, která má střed shodný se středem objektu [2].



(a) Kužel (b) Úsečka (c) Koule

(d) Dveře (e) Značka

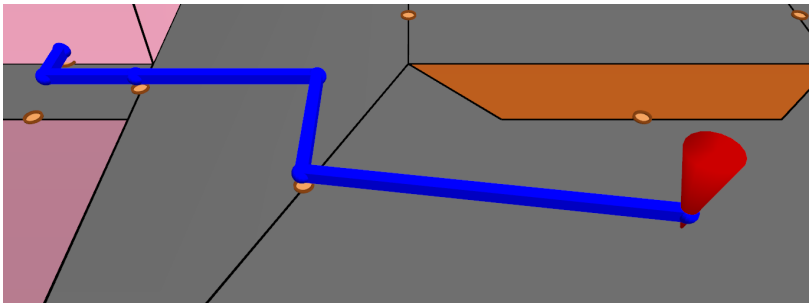
Obrázek 5.7: Vybrané popisované modely

5.3 Zobrazení trasy ve 3D modelu budovy

Trasu vyhledané cesty můžeme vnímat jako lomenou čáru, která prochází určitými body. Model trasy je tedy množina úseček od výchozího bodu po cílový bod. Cílovým bodem jsou jedny z dveří požadované místnosti.

Hlavním bodem trasy je bod *zde stojíte*, je to výchozí bod trasy a měl by být na mapě hned vidět. Vyobrazením počátečního bodu je červený kužel otočený podstavou nahoru. Jak je vidět na obrázku 5.8, červená barva přitahuje pozornost a modrá barva trasy kontrastuje s šedou barvou chodeb a jsou tedy oba modely dobře rozeznatelné od okolí.

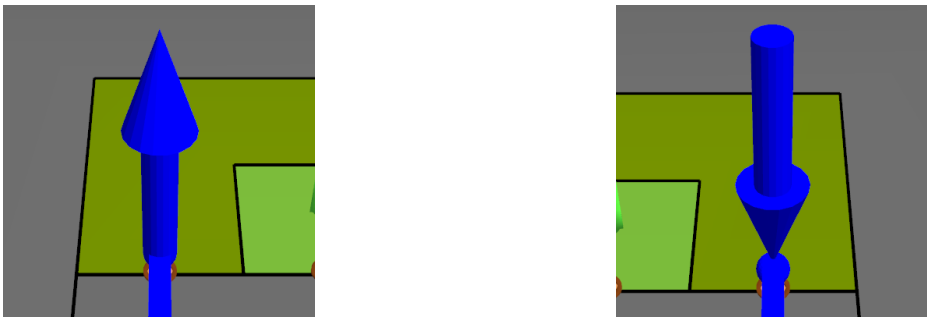
Jak již bylo zmíněno, trasa prochází body. Tyto body dělíme na výchozí bod (*zde stojíte*), průchozí body, dveře a východy. Každý bod má svoje chování a vzhled. Výchozí bod určuje místo, kde se bude nacházet červený kužel. Dveře spojují místnosti a jsou vyobrazeny hnědými kruhy na hranách místností. Průchozí body zobrazovány nejsou, ale spojují



Obrázek 5.8: Vyhledaná trasa

jednotlivé dveře, a východy určují, kde budova končí a kudy může uživatel odejít. Průchozí body jsou vyobrazeny až v případě modelu trasy jako sférické spoje jednotlivých linek trasy.

Specifickou vlastností trasy je rozdělení jejího modelu do částí v případě, že je vyhledávání prováděno skrze patra. Jednotlivé části náleží patrům, kterými trasa prochází. Je nutné znázornit, že trasa mění patro. Nejlogičtějším vyobrazením změny patra trasy je šipka nahoru či dolů v posledním bodě trasy aktuálního patra, viz obrázek 5.9.



Obrázek 5.9: Šipky změny patra pro vyhledané trasy

5.4 Demo režim aplikace

Tzv. demo režim je stav, do kterého aplikace přejde po určité době nečinnosti. Principem je procházení místností na patrech budovy v podobě náhodného postupného označování místností. Demo režim se spouští po určité době od poslední interakce uživatele a jednotlivá označení místností trvají také určitou dobu. Tyto časové údaje jsou čerpány z konfiguračního souboru.

Implementace demo režimu je uložena v projektu BuildingViewer.ViewModels, protože je přímo navázána na zobrazovací View model VisualizerViewModel. Demo režim sestává z dvou časovačů, jednoho spouštěcího a druhého, který náhodně vybírá a označuje místnosti. Jakmile spouštěcí časovač dojde k události tiku, sám sebe zastaví a spustí časovač demo režimu. Časovač demo režimu je ukončen okamžitě po interakci uživatele s aplikací. Spolu s ukončením je vrácena mapa do původního zobrazení, tedy do patra s bodem *zde stojíte*. Demo režim prochází všechna patra postupně s rostoucí tendencí, začíná vždy v patře výchozím.

5.5 Grafické uživatelské rozhraní aplikace

Front-end aplikace je rozdělen do tří hlavních panelů, v prostředním panelu se nachází Viewport3D, nalevo ListBox s patry budovy a napravo hlavní ovládací a informační prvek pravý panel reprezentovaný komponentou ContentControl. Viewport3D a seznam pater budovy jsou obaleny jednou UserControl komponentou s názvem VisualizerUserControl. Pravý panel je navržen tak, aby se jeho obsah měnil podle stavu aplikace, ale také umožňoval tento stav měnit, resp. vracet se do historie.

5.5.1 Visualizer

Veškeré zobrazování mapy je prováděno zde, ve Visualizeru, v komponentě UserControl, která obsahuje Viewport3D a ListBox pro seznam pater. Visualizer je navázán na VisualizerViewModel, který obsahuje hlavní funkčnost aplikace. Probíhá zde načítání dat z databáze a jejich převod do 3D modelů. Pomocí DataBindingu je obousměrně navázaný levý ListBox na vlastnost *aktuální patro* (CurrentFloor).

5.5.2 Pravý panel

Pravý panel je poměrně obsáhlou a složitou komponentou aplikace, reaguje na stav aplikace přepínáním karet a ukládá historii stavů. Je navázán na MainViewModel, se kterým primárně komunikuje. Vlastnost State určuje aktuálně zobrazenou kartu, kdy je obsah pomocí DataTemplateSelectoru konvertován na určitou kartu. Každá karta má svůj UserControl, který je taktéž navázaný na MainViewModel. Je tím zajištěna vazba na stejná data a není porušen MVVM.

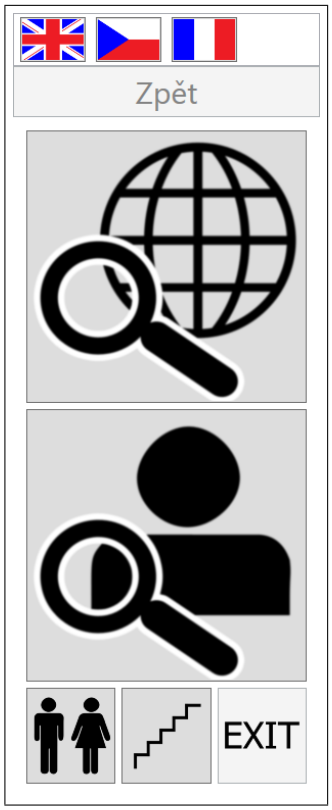
Historie stavů je uložena ve formě zásobníku v MainViewModelu. Vkládání na zásobník je prováděno pouze tehdy, pokud uživatel něco změní nebo vykoná určitou operaci. Aplikace sama na zásobník nevkládá (např. v demo režimu by s každou označenou místností byl přidán záznam do historie). Do historie se vkládají tyto informace o stavu aplikace:

- aktuálně označená místnost a patro
- vyhledaná trasa
- vybraná osoba
- vyhledávaný text místností a osob
- obsah vlastnosti State

Díky všem těmto hodnotám je možné se vracet v historii vybraných místností, osob apod. Návrat k předchozímu stavu je prováděn pomocí tlačítka Zpět a znamená to odebrání záznamu ze zásobníku a nastavení obsahu vrcholu zásobníku do aplikace. Vrchol zásobníku historie tedy reprezentuje aktuální stav aplikace.

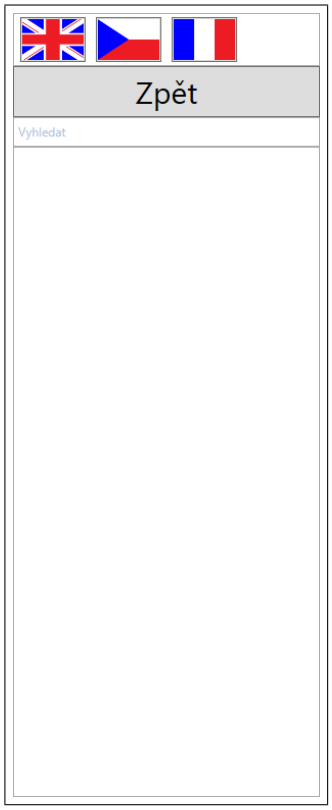
Při vracení se bylo nutné zajistit, aby aplikace všechny změny zobrazila. Například změna místnosti na rozdílných patrech musí nejprve změnit patro a následně vysunout místnost.

Jednotlivé karty pravého panelu mají svůj konkrétní učel, každá zobrazuje určité informace. Na následujících obrázcích jsou demonstrovány jednotlivé karty spolu s vysvětlením, co zobrazují a jaké funkce obsahují.



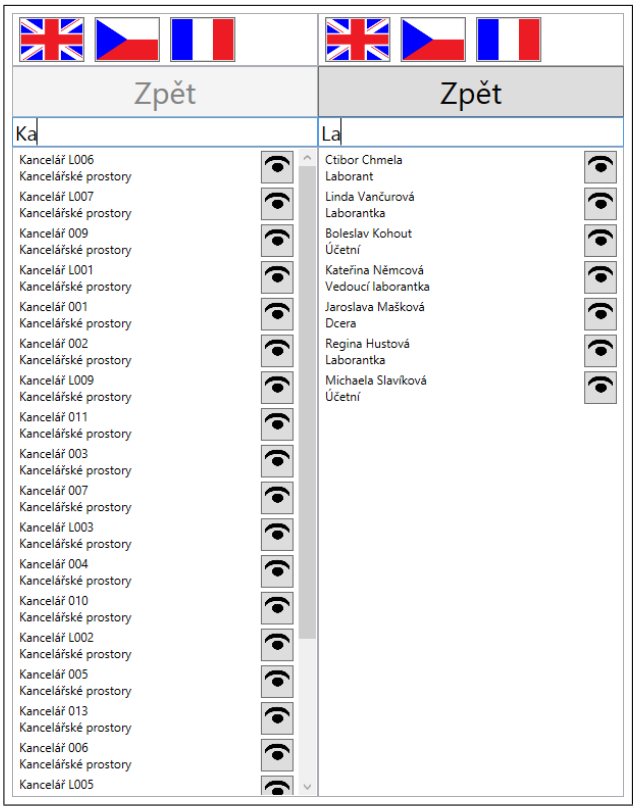
Hlavní menu aplikace obsahuje dvě hlavní tlačítka, *vyhledávat místnost* a *vyhledávat osobu*. Pod těmito tlačítky se nachází rychlé volby, vyhledat toalety, schodiště nebo východy.
Východy jsou prozatím znepřístupněny, protože je předpokládáno, že se panel nachází u východu a není tedy nutno vyhledávat k tomuto východu trasu. Tato možnost je tu připravena pro situace, kdy u určitého východu je vchod na parkoviště apod.

Obrázek 5.10: Karta hlavní menu



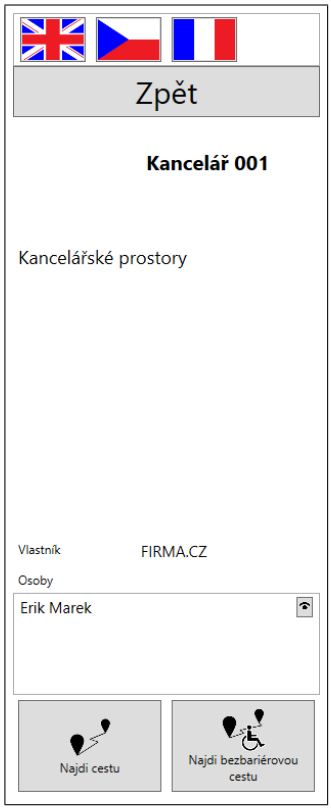
Vyhledávání místností a osob poskytuje uživateli zadání textového řetězce pro vyhledání místnosti. Text je vyhledáván v reálném čase a výsledky jsou zobrazovány do seznamu pod vyhledávacím polem.
Zobrazena je pouze jedna karta, protože obě karty mají stejné rozložení komponent, pouze jinak reagují, viz následující dvojice karet.

Obrázek 5.11: Karty vyhledávání místností a osob



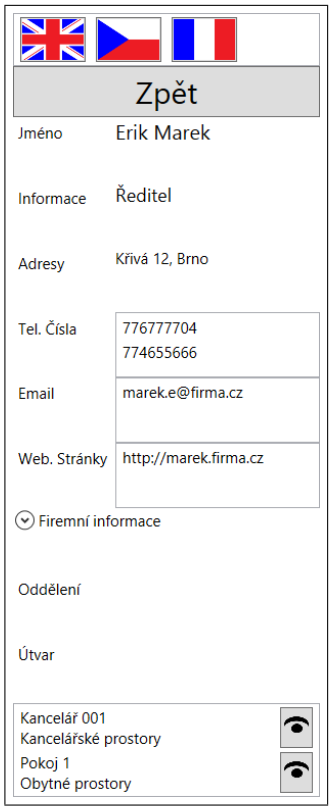
Výsledky vyhledávání uživateli zobrazují seznam s odpovídajícími nálezy. Obě karty umožňují u jednotlivých položek seznamu přistoupit k detailu, a to pomocí tlačítka v pravé části položky. Karta je pak přesměrována na detailní kartu místnosti nebo osoby.

Obrázek 5.12: Karty s výsledky vyhledávání



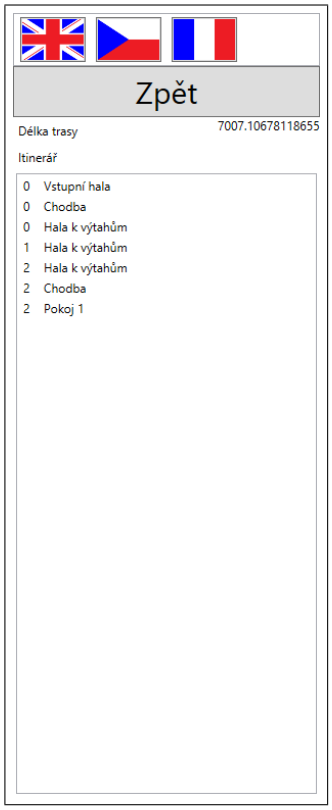
Detail místnosti zobrazuje data známá o konkrétní místnosti, která mohou být zobrazena uživateli. Karta umožňuje spuštění vyhledávání standardní nebo bezbarierové trasy a nebo přístup k detailu osoby, která se v místnosti vyskytuje.

Obrázek 5.13: Karta detail místnosti



Detail osoby vypisuje do tabulky veškerá data známá o vybrané osobě. Ve spodní části se nachází seznam místností svázaných s touto osobou. Opět každá položka obsahuje tlačítko pro přístup k místnosti.

Obrázek 5.14: Karta detail osoby



Vyhledaná trasa zatím obsahuje data o délce trasy a itinerář trasy. Každá položka itineráře sestává z dvou hodnot, čísla podlaží a názvu místnosti, kterými je nutné projít. Současně s touto kartou je vyobrazena trasa. Do budoucna je počítáno s implementací interaktivity itineráře, pro názornější procházení trasy.

Obrázek 5.15: Karta vyhledané trasy

Kapitola 6

Vyhledávání trasy

Podstatnou funkcí aplikace je možnost vyhledávat trasu v budově. Cílovými objekty vyhledávání jsou místnosti. Každá z místností má své dveře, kterými je možné vejít dovnitř. Těchto dveří však může být více a místnost tedy může být průchozí. Jestli je místnost průchozí je možné ovlivnit také parametrem IsWalkthrough rovnou v databázové tabulce.

Do vyhledávání se však počítají i metody vyhledání místností či osob. Jelikož jsou všechny místnosti i osoby načteny v paměti od startu (příp. od stažení aktualizací), můžeme vyhledávat konkrétní řetězec napřímo. Je k tomu využita funkce Contains z knihovny String. Vyhledává se pouze v položkách Title a Description u místností, a Name spolu s Description u osoby. Obsahy těchto položek jsou čistě textové a nijak dlouhé, je tedy možné nad nimi vyhledávat pouze pomocí funkce Contains. Pokud by bylo vyhledávání prováděno nad rozsáhlejšími texty, bude nutné dodatečně implementovat funkce jako např. Knuth-Morris-Pratt algoritmus [6].

6.1 Vytvoření grafu stavového prostoru

V databázové sekci je popsán způsob, jakým jsou data o grafu uložena v databázi. Jedná se o dvě podstatné tabulky Node a NodeEdges. Node zastupují uzly a NodeEdges hrany grafu.

6.1.1 Metody rozmísťování bodů

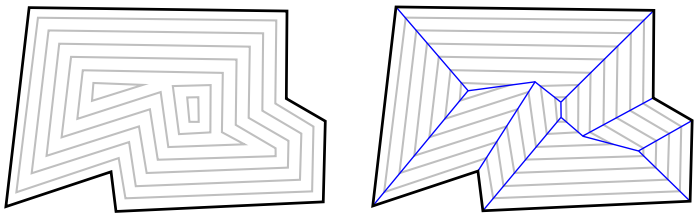
Během vývoje došlo k několika změnám ohledně způsobu rozmístění průchozích bodů uvnitř polygonu místnosti. Uvažovalo se nad automatickým rozmístěním bodů při vytváření polygonu a následném spojení jednotlivých bodů hranami.

Osy místností Jednalo se jednu z prvních možností, jak vytvořit body uvnitř polygonu místnosti. Cílem algoritmu bylo v každém z rohů vytvořit osu. Následně najít průsečíky těchto os, které jsou uvnitř polygonu. Mezi těmito průsečíky následně vytvořit spojnice. Tímto postupem získáme pomyslnou osu místnosti. Všechny dveře pak připojíme k této ose spojnicí k nejbližšímu průsečíku. Pokud však by úhel byl větší jak 45°, vytvoří se kolmice na stěnu se dveřmi a spojnice se vytvoří mezi dveřmi a průsečíkem kolmice se spojnicí osy místnosti. Bohužel se ukázalo, že algoritmus nevytváří osu místnosti korektně.

Straight skeleton Tento algoritmus se používá k výpočtu zastřešení budovy a střešních štítů, což v podstatě odpovídá požadavkům algoritmu pro získání kostry (osy) místnosti.

Pracuje částečně na stejném principu jako předchozí algoritmus. Jako první najde osy všech vnitřních úhlů. Následně polygon proporcionálně zmenšuje po těchto osách, dokud se nepotkají protilehlé stěny. Výstupem algoritmu je kostra polygonu složená z hlavní osy polygonu a ze spojnic utvořených původními osami vnitřních úhlů ke krajním bodům.

Graf je nutné následně doplnit o spojnice ke dveřím, to je možné provést vyvedením kolmic z bodu dveří až po průsečík s kteroukoli spojnicí dosavadní kostry.

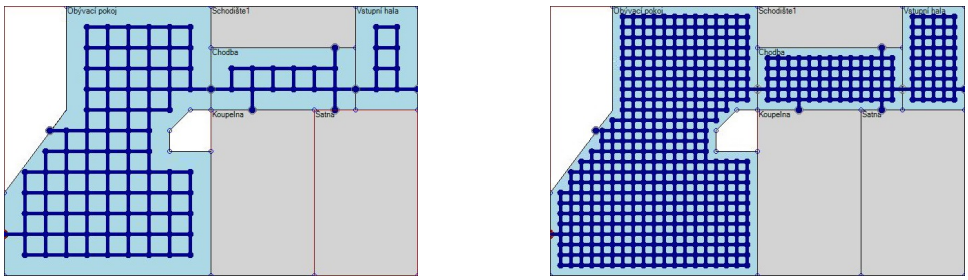


Obrázek 6.1: Postup algoritmu Straight Skeleton

Proložení mřížkou Dalším z algoritmů pro vložení bodů do plochy polygonu bylo proložení polygonu konstantní mřížkou. Rozmísťování bodů bylo prováděno po řádcích a body byly vkládány s určitým rozestupem. Pro určení, zda-li se bod nachází uvnitř plochy polygonu, byla použita funkce Ray Casting. Jedná se o funkci, která postupuje od výchozího bodu jedním směrem a s každým protnutím jakékoliv hrany polygonu inkrementuje vnitřní počítadlo. Funkce se zastaví, až dorazí k hledanému bodu a nebo překročí hranice. Jestli je bod uvnitř, funkce určuje podle hodnoty počítadla. Pokud je sudé, bod je vně polygonu, pokud je liché, bod je uvnitř. Počítadlo musí být inicializováno na hodnotu 0. V tomto případě byla funkce volána vždy s Y souřadnicí shodnou s hledaným bodem a X souřadnicí odpovídající X souřadnici nejlevějšího bodu polygonu. Bylo tím docíleno, že algoritmus vždy dorazí k hledanému bodu a vykoná nejméně iterací.

Jelikož C# nemá třídu pro Polygon, pro tento algoritmus byla vytvořena i s vnitřními funkcemi, které vytvářely seznamy hran polygonu pro budoucí vyhledávání bodů uvnitř.

Nakonec se algoritmus ukázal jako zbytečně paměťově náročný z pohledu množství výsledných bodů a spojnic na jeden polygon. Ve výsledku zpomaloval i výsledné vyhledávání, kvůli velkému množství prohledávaných bodů.



Obrázek 6.2: Rozmístění bodů v místnostech s různými hustotami mřížek na demonstračních místnostech

Pevně rozmístěné body Nejjednodušším ale nejpracnějším zbůsobem je ruční zadání bodů a jejich spojení do hran pomocí editační aplikace. Body jsou vkládány tak, aby byly

v určité vzdálenosti od dveří směrem do chodby nebo průchodzích koridorů. Spojnice mezi nimi pak vytvoří jednoduchou síť cest.

Editace bodů a hran je činnost výhradně určená administrátorovi nebo správci budovy, ten může trasu různě upravovat i podle aktuálního stavu. Jedná se pravděpodobně o nejjednodušší způsob, jak vytvářet hrany s parametry, konkrétně bezbariérové trasy. U předchozích algoritmů, bylo nutné všechny hrany dohledat zpětně a upravit jejich parametry.

6.1.2 Využití parametrů hran

Aktuální stav vyhledávání respektuje dva parametry hran, bezbarierovost a počet schodů. Tyto dva parametry spolu logicky souvisí, ale nemusí vylučovat jeden druhý. Pokud bude zadaný počet schodů znamená to, že hrana není bezbarierová, např. schodiště. Ale konkrétně schodiště mohou být opatřena plošinou pro vozíčkáře, je tedy možné nastavit parametr bezbarierovosti hrany.

Z počtu schodů je odvozována délka trasy jdoucí přes tuto hranu. Je vycházeno z norem o schodištích[14], konkrétně z částí o nášlapných plochách a výškách schodu. Lambertovo pravidlo definuje vztah mezi výškou schodu a šířkou nášlapnice jako:

$$2h + b = 630\text{mm}$$

kde h je výška a b je šířka schodišťového stupně. Akceptován je rozsah výsledných hodnot 600mm – 650mm. Nejmenší šířka stupnice je 250mm, pak je výška schodišťového stupně od 175mm po 200mm v akceptovaném rozsahu. V implementaci této části je uvažováno nad 200mm jako výškou schodu a 250mm jako šířkou stupnice. Pak se Pythagorovou větou odvodí délka trasy na jeden schod a tato hodnota se vynásobí počtem schodů z parametru hrany. Po zaokrouhlení se jedná o hodnotu 320mm na jeden schod.

6.2 Algoritmy

K vyhledávání trasy uvnitř budovy je používáno množství algoritmů, podle typu místnosti. Algoritmy vycházejí z předpokladů, že se v místnosti nacházejí překážky v podobě nábytku apod., nebo jsou místnosti prázdné, či je nábytek ignorován.

Implementace algoritmů byly převzány a přepsány do C# z přednášek předmětu Základy umělé inteligence. Jedná se konkrétně o algoritmus UCS popsaný níže.

6.2.1 Door-to-door algoritmus

Jedním z algoritmů akceptujících nábytek a překážky uvnitř místností je například Door-to-door algoritmus[13]. Tento algoritmus slouží k vyhledání cesty pro pěší z výchozí místnosti do cílové místnosti skrze dveře. Tyto dveře jsou mezi sebou buď viditelné a nebo viditelnosti brání překážka. Algoritmus obejde tuto překážku nejkratší možnou cestou, tedy přímo přes roh, což znemožňuje výsledné zobrazení trasy uživateli bez patřičných úprav.

Door-to-door algoritmus je závislý na následujících předpokladech:

1. Sémanticko-geometrický model budovy je známý a je rozhodující pro automatizaci vyhledávání trasy, vzhledem k tomu, že poskytuje informace o umístění, typu a stavu dveří (zamčené nebo odemčené).

2. Model poskytuje přímo, nebo nepřímo informace o propojení místností.

3. Model poskytuje informace o všech vnitřních objektech, které se v určitém okamžiku mohou stát překážkou.

4. Dynamické změny itineráře a struktury budovy jsou taktéž známé, aby na ně mohl algoritmus zareagovat a přepočítat trasu.

Bod 3 není splnitelný v této práci, protože vnitřní uspořádání místností v podstatě nemá vliv na základní orientaci v budově, a tak tyto data nejsou obsažena v databázi.

Tento algoritmus se pro použití v tomto projektu z počátku zdál jako nejvhodnější, ale z důvodu komplikovanějších modifikací pro lepší zobrazení trasy od něj bylo upuštěno a dále byl brán spíše jako inspirace.

6.2.2 A* algoritmus

Nejvhodnějším algoritmem byl shledán algoritmus A*, jakožto nejpoužívanější algoritmus v oblasti vyhledávání cest v kladně ohodnocených grafech. Vychází z Dijsktrova algoritmu[1] a přidává heuristickou funkci. Využívá princip hladovění z počátečního bodu do cílového, výslednou trasou je tedy ta s nejlepším ohodnocením.

A* využívá funkci $f(x)$ ohodnocující jednotlivé uzly. Ta je dále složená z funkcí $g(x)$ a $h(x)$, kdy $g(x)$ představuje cenu překonání vzdálenosti mezi počátečním a aktuálním uzlem a $h(x)$ je heuristickou funkcí. Tato funkce odhaduje optimálnost postupu cesty za pomoci ceny překonání vzdálenosti z aktuálního uzlu do cílového uzlu a zároveň nemůže nadhodnocovat vzdálenost k cíli.

Z algoritmu bylo upuštěno kvůli jeho implementační náročnosti oproti nízkému počtu procházených bodů. Jelikož se jedná o opravdu malý počet uzlů v grafu, je s algoritmem počítáno spíš do budoucna jako s možnou alternativou.

6.2.3 Uniform-cost search algoritmus

Jako nejvhodnější algoritmus pro implementaci vyhledávání v projektu byl vybrán algoritmus Uniform-cost search (zkráceně UCS). Tento algoritmus vychází z algoritmu prohledávání do šířky (Breadth-first search), přidává však funkci určující následující bod k expandování. Nejčastěji se jedná o bod s nejkratší cestou od aktuálního bodu. UCS algoritmus má 2 verze, bez seznamu CLOSED a se seznamem CLOSED. V programu je použita modifikace algoritmu UCS se seznamem CLOSED.

Algoritmus má následující postup:

1. Vytvoř dva seznamy OPEN a CLOSED, do seznamu OPEN vlož počáteční uzel včetně ohodnocení.

2. Pokud je seznam OPEN prázdný, ukonči vyhledávání jako neúspěšné, jinak pokračuj.

3. Vyber prvek ze seznamu OPEN s nejlepším ohodnocením a ulož ho do seznamu CLOSED.

4. Jedná-li se o cílový prvek, vrať cestu od kořenového prvku k cílovému prvku, jinak pokračuj.

5. Do seznamu OPEN ulož všechny bezprostřední následovníky, kteří nejsou ani v seznamu OPEN ani v seznamu CLOSED.

6. Opakuj bod 2.

Ohodnocující funkce prošla během vývoje několika verzemi. První verze funkce byla v jejím původním znění z definice UCS algoritmu, tedy nejkratší cesta k následujícímu. Druhou verzí, dlouho testovanou, bylo vybírání bodu s nejkratší cestou k cílovému bodu. U této verze bylo výrazně urychleno vyhledávání v rámci jednoho patra, problém však nastal při vyhledávání na více patrech. Jelikož funkce k výpočtu vzdálenosti určovala vzdálenost dvou bodů v ploše, nebyla akceptována změna patra jako mnohdy velký nárůst vzdálenosti[1]. Třetí a aktuální verze funkce počítá se změnami pater a celkově upravuje styl vyhledávání. Určování následujícího bodu dělí do dvou větví, pokud se aktuální bod nachází na stejném patře jako cíl, je vrácena plošná vzdálenost mezi těmito body. Pokud však nejsou na stejném patře, je vrácena vzdálenost k výtahům či schodištím. Jelikož se jedná pouze o funkci určující ohodnocení bodů, algoritmus UCS není nijak pozměněn.

6.3 Implementace vyhledávání

Vyhledávací grafy využívají pojmy uzel, hrana a spojnice. Pro tyto pojmy existují programové reprezentace, se kterými pak algoritmy pracují. V případě uzlu se jedná o třídu Node, hranou a zároveň spojnicí je pak třída NodeEdge. Konkrétně třída Node se však v implementaci algoritmů nedá použít, protože obsahuje pouze informace o uzlu. Byla proto navržena obalová třída PathfindingNode, obsahující veškeré potřebné prostředky pro realizaci vyhledávání.

Obalová třída PathfindingNode slouží k rozšíření funkčnosti obecné třídy Node popisující body mapy, jako jsou dveře, východy a průchozí body na chodbách a v místnostech. Rozšiřuje třídu Node o položky potřebné pro vyhledávání, jako je fronta uzlů od počátku, délka trasy od počátku, apod.

Třída Pathfinder sdružuje všechny parametry k provedení vyhledávání. Obsahuje vyhledávací metodu, určenou typem vyhledávané trasy, a parametry vyhledávání, konkrétně přepínač vyhledávání bezbariérové cesty. Pro spuštění vyhledávání je nutné vytvořit novou instanci této třídy a zadat cílovou destinaci v podobě místnosti.

6.3.1 Vyhledávací metody

Do budoucna je předpokládáno více implementovaných vyhledávacích metod, kvůli rozdílným podmínkám, např. vyhledávání trasy vně budovy v areálu. Byla proto navržena abstraktní bázová třída BaseMethod implementující rozhraní IBaseMethod. Obsahem této třídy je především BackgroundWorker, který v novém vlákně provádí samotné vyhledávání. Procedura DoWork, která je označena za abstraktní, musí být implementována všemi dědícími třídami. Vyhledávání je spouštěno funkcí Run(). Po skončení práce BackgroundWorker třídy je volán event PathFound, pouze pokud byla trasa nalezena.

Implementovanou metodou je Uniform-cost search v třídě UniformCost a do budoucna je počítáno s metodou Bidirectional search, právě pro případy vyhledávání mezi patry.

[1] S každým patrem se relativní vzdálenost trasy zvětšuje. Konkrétně schodiště do trasy vkládají celé úseky.

Kapitola 7

Závěr

V této práci jsem se zaměřil na implementaci aplikace pro zobrazení mapy ve 3D, která umožňuje vyhledávat trasu. Práce obsahuje popis stávajících řešení, která v přípravné fázi práce posloužila jako inspirace, ale i jako poučení. Primárně se práce zabývá právě implementací prezentační a vyhledávací části, jsou popsány algoritmy, kterými byly vytvářeny 3D modely místností budovy, ale i model trasy apod.

Výsledkem práce je funkční aplikace, která čerpá data z databáze a zobrazuje je ve formě mapy ve 3D, umožňuje vyhledávání trasy v rámci celé budovy. Aplikace je modifikovatelná z konfiguračního souboru a pracuje jako klientská aplikace vůči databázovému serveru. Aplikací může běžet zároveň více a každá reprezentuje data v databázi s lokálními úpravami specifickými pro konkrétní instanci aplikace. Po určité době nečinnosti uživatele aplikace přechází do demo režimu, kdy náhodně označuje místnosti a zobrazuje jejich detaily.

Testování aplikace probíhalo na 4 zařízeních s operačními systémy Windows 8.1 a 10, jedno zařízení byl dotykový 10 palcový tablet s Windows 10 a ostatní notebooky a jeden stolní PC. Testovacím serverem byl jeden z notebooků, na kterém běžel SQL Server 2008 s databází naplněnou daty imaginární kancelářské budovy.

V budoucnu hodlám aplikaci dále vyvíjet a přizpůsobovat požadavkům možných realizací. Je více než nutné implementovat administrátorskou aplikaci, pro správu dat databáze i aplikace. Z hardwarové stránky aplikace potřebuje dostatečně velký dotykový monitor a odpovídající stojan.

Literatura

[1] CORMEN, Thomas H.: *Introduction to algorithms*. 2nd ed. Cambridge, Mass.: MIT Press, 2001, ISBN 0-07-013151-1.

[2] DELOURA, Mark: *Game programming gems*. Hingham: Charles River Media, 2000, ISBN 1-58450-049-2.

[3] Eberly, D.: Triangulation by Ear Clipping [online]. `http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf`, 2002 [cit. 2016-01-21].

[4] GARY MCLEAN HALL: *Pro WPF and Silverlight MVVM effective application development with Model-View-ViewModel*. New York: Apress, 2010, ISBN 978-1-4302-3163-9.

[5] JAMES, Buddy a Lori LALONDE: *Pro XAML with C#*. Berkley, CA: Apress, 2015, ISBN 978-1-4302-6775-8.

[6] KNUTH, Donald E., James H. MORRIS a Vaughan R. PRATT: Fast Pattern Matching in Strings. *SIAM J. Comput.*, ročník 6, č. 2, 1977: s. 323–350.

[7] NOBLE, Sam, Sam BOURTON a Allen JONES: *WPF recipes in C# 2008: a problem-solution approach*. New York: Distributed to the book trade worldwide by Springer-Verlag New York, 2008, ISBN 978-1-4302-1084-9.

[8] WWW stránky: Overview of the .NET Framework. `https://msdn.microsoft.com/en-us/library/zw4w595w.aspx`, [cit. 2015-12-29].

[9] WWW stránky: WPF Windows Overview. `https://msdn.microsoft.com/library/ms748948.aspx`, [cit. 2015-12-29].

[10] WWW stránky: 3D Modeling & Rendering Software. `http://www.autodesk.com/products/3ds-max/overview`, [cit. 2016-02-10].

[11] WWW stránky: C# Helper. `http://csharphelper.com`, [cit. 2016-02-11].

[12] WWW stránky: Entity framework. `https://entityframework.codeplex.com`, [cit. 2016-02-21].

[13] ZLATANOVA, S. a L. LIU: A "Door-to-door"Path-finding approach for indoor navigation. , č. 1, 2011: s. 1–6.

[14] ČSN 73 4130: Schodiště a šikmé rampy – Základní požadavky. Norma, Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, Praha, CZ, 2010.