VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

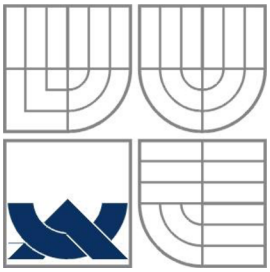
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU 3D - HLEDÁNÍ CESTY V BUDOVĚ

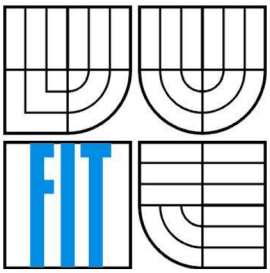
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Ján Macek
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU 3D - HLEDÁNÍ CESTY V BUDOVĚ

3D PATHFINDING - SEARCHING PATH IN A BUILDING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Ján Macek
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jan Samek, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Předmětem této bakalářské práce je popis algoritmů sloužících na prohledávání stavového prostoru a následná implementace aplikace pro hledání cesty v budově. První část práce se věnuje metodám popisu prostoru. V další kapitole jsou vysvětleny metody vyhledávání cesty v prostoru. Třetí část práce popisuje postup implementace aplikace. Na závěr jsou uvedeny výsledky testování a jejich zhodnocení.

Abstract

The subject of this thesis is the description of the algorithms used for searching in the state space and the subsequent implementation of a searching path for the building. The first part of the thesis is devoted to methods for area description. The next chapter explains the methods of pathfinding in space. The third part of the paper describes the implementation of an application. In conclusion, are the results of testing and their assessment.

Klíčová slova

Kartézská soustava souřadnic, grid, Delaunayova triangulace, stavový prostor, vyhledávaní cesty v 3D, A* algoritmus, hierarchický A* algoritmus, algoritmus „od dveří ke dveřím“.

Keywords

Cartesian coordinate system, grid, Delaunay triangulation, state space, 3D pathfinding, A * algorithm, hierarchical A * algorithm, "door to door" algorithm.

Citace

Macek Ján: Prohledávání stavového prostoru 3D - hledání cesty v budově, bakalářská práce, Brno, FIT VUT v Brně, 2013

Prohledávání stavového prostoru 3D - hledání cesty v budově

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing.Jana Samka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ján Macek
10. 5. 2013

Poděkování

Rád bych touto cestou poděkoval těm, kteří mi při přípravě této práce podali pomocnou ruku. Zvláštní poděkování patří vedoucímu práce Ing. Janu Samkovi, Ph.D. za jeho odborné vedení při vypracovávání práce, pomoc poskytnutou při řešení problému a pravidelné konzultace, což vedlo k úspěšnému dokončení bakalářské práce.

Obsah

1 Úvod

Ľudstvo sa v dnešnej dobe snaží dostať kam potrebuje v čo najkratšom čase a bez zbytočných zdržaní a obchádzok. Čas je veľmi dôležitý a preto hľadajú najlepšie cesty ako sa presunúť do miesta ich záujmu. Z tohto dôvodu sa objavujú zariadenia, ktoré takúto trasu dokážu vyhľadať a predložiť ju užívateľovi. Keď sa na túto problematiku pozrieme z iného uhla, vyhľadávanie trasy medzi dvomi bodmi nemusí nutne slúžiť priamo pre potreby ľudí. Túto technológiu môže využívať umelá inteligencia v robotike alebo v software, ako sú napríklad počítačové hry s 3D prostredím.

Tému vyhľadávanie cesty v priestore, respektíve tému prehľadávania stavového priestoru som si vybral, pretože oblasť umelej inteligencie je pre mňa obzvlášť zaujímavá. Nielen to, že stroj dokáže vykonávať činnosť, na ktorú bol vyrobený a to rýchlejšie ako jeho vyhotoviteľ, ale dokáže sa v priebehu svojho života aj učiť. A práve spomínaná rýchlosť, s ktorou umelá inteligencia v dnešnej dobe pracuje, podnecuje vo vývoji spôsobov výpočtu, ktoré sú časovo optimálnejšie, pamäťovo menej náročné a dokážu rýchlosť, ktorou problém vyhodnocujú, ešte znásobiť.

V dobe, keď som študoval materiály, ktoré obsahovali informácie o tejto problematike som narazil na články, ktoré poukazovali na stále používanie algoritmov, ktoré sú zastarané a na to, že existujú algoritmy časovo a pamäťovo výhodnejšie. Práve preto som sa zameral na algoritmy, ktoré spĺňajú nároky nižšej časovej náročnosti . Za cieľ tejto práce som si stanovil priblíženie metód popisu priestoru, ktoré sú vhodné na prehľadávanie, vysvetlenie fungovania algoritmov pre vyhľadávanie cesty v priestore a predloženie získaných vyhodnotení algoritmov, ktoré budú vyplývať z ich implementácie a testovania.

V druhej kapitole bakalárskej práce sa zaoberám metódami popisu priestoru. Metódami ktoré popisujú priestor reálneho sveta a tými, ktoré uchovávajú informácie o priestore tak, aby ich bolo možné systematicky prehľadávať.

Tretia kapitola približuje samotné algoritmy ktoré vyhľadávajú trasu v priestore medzi zvolenými bodmi. Na začiatku sú popísané algoritmy ktoré vyhodnocujú dvojrozmerný priestor. Následne sú rozobraté princípy vyhľadávania cesty v trojrozmernom priestore.

Implementácia aplikácie, ktorá bola hlavným výstupom tejto práce je vysvetlená vo štvrtej kapitole. Podrobne tu popisujem jej základné aspekty, ktorými sú reprezentácia budovy a jej následná vizualizácia a implementácia vyhľadávacích algoritmov.

Piata kapitola je zameraná na porovnanie výsledkov implementovaných algoritmov v rôznych prípadoch jej použitia.

Záver práce, šiesta kapitola, diskutuje dosiahnuté výsledky, možné využitie a rozšírenia mojej aplikácie.

2 Metódy popisu priestoru

Aplikácie, ktoré pracujú s priestorom, potrebujú mať definované pravidlá, podľa ktorých je daný priestor popisovaný. Bez týchto pravidiel by aplikácia nevedela, ako pristupovať k dátam priestoru a ako ich spracovať pre svoje potreby. V nasledujúcich kapitolách budú predstavené metódy popisu priestoru, ktoré sú vhodné na použitie pri riešení zadania bakalárskej práce.

2.1 Metódy popisu priestoru reálneho sveta

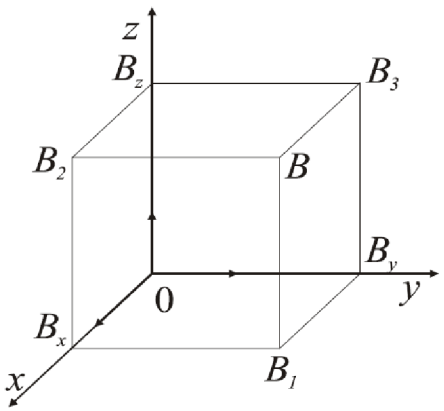
Predmety reálneho sveta a priestor v ktorom sa nachádzajú je vhodné popisovať pomocou súradníc. Keďže ide o popis trojrozmerného priestoru aj súradnice musia mať tri rozmery, tými sú šírka, hrúbka a výška.

2.1.1 Karteziánska sústava súradníc

Na popis priestoru je najčastejšie používaná pravotočivá trojrozmerná karteziánska sústava súradníc [1]. Pravouhlý súradnicový trojhran je útvar pozostávajúci z bodu *O*, z troch priamok *x,y,z*, ktoré prechádzajú bodom *O* pričom sú navzájom kolmé a z troch rovín $\pi = (x, y), v = (x, z), \mu = (y, z)$. Bod *O* nazývame začiatkom, priamky *x,y,z* osami a roviny π, v, μ, rovinami trojhranu.

V Euklidovskom priestore E_3 najčastejšie používame karteziánsku súradnicovú sústavu so začiatkom *O*, s osami *x,y,z* a jednotkovými vektormi $\vec{x}, \vec{y}, \vec{z}$. Je to bijektívne zobrazenie, ktoré každému bodu priradí usporiadanú trojicu reálnych čísel [x, y, z]. Označenie karteziánskej súradnicovej sústavy je potom (*O*, x, y, z).

B_1 - kolmý priemet bodu *B* do $\pi = (x, y)$
B_2 - kolmý priemet bodu *B* do $v = (x, z)$
B_3 - kolmý priemet bodu *B* do $\mu = (y, z)$
$|BB_3|$ - *x*-ová súradnica bodu *B*
$|BB_2|$ - *y*-ová súradnica bodu *B*
$|BB_1|$ - *z*-ová súradnica bodu *B*

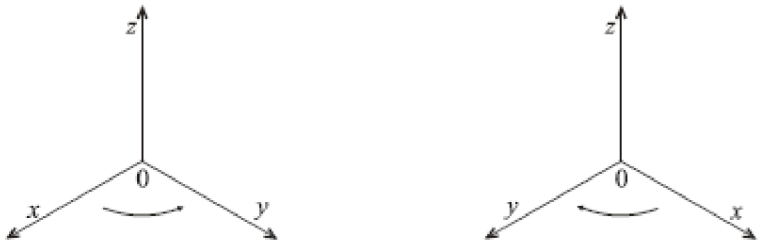


Obr.1 Model karteziánskej sústavy súradníc [1].

Pre vzdialenosť dvoch bodov v priestore platí: Nech A = [x_1, y_1, z_1] a bod B = [x_2, y_2, z_2] potom pod vzdialenosťou bodu *A* od bodu *B* rozumieme:

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Hovoríme, že súradnicová sústava je pravotočivá (ľavotočivá) ak usporiadaná trojica vektorov $\vec{x}, \vec{y}, \vec{z}$, je pravotočivá (ľavotočivá) .



Obr.2 Pravotočivá a ľavotočivá súradnicová sústava [1].

2.1.2 Mriežková reprezentácia (Grid)

Pri potrebe prehľadávať obsiahly priestor, ktorý je reprezentovaný karteziánskou sústavou súradníc, by bola časová náročnosť príliš vysoká a preto je potreba priestor zjednodušiť a tým znížiť jeho náročnosť.

Mriežka delí priestor do malých pravidelných obrazcov, ktoré sú nazývané dlaždice. Tieto dlaždice môžu byť tvaru štvorca, trojuholníku alebo šesťhranného polygónu. Spracovávaný priestor je teda rozdelený na dlaždice a následne sú vytvorené prepojenia medzi nimi ktoré reprezentujú ich spojenie do jedného celku.

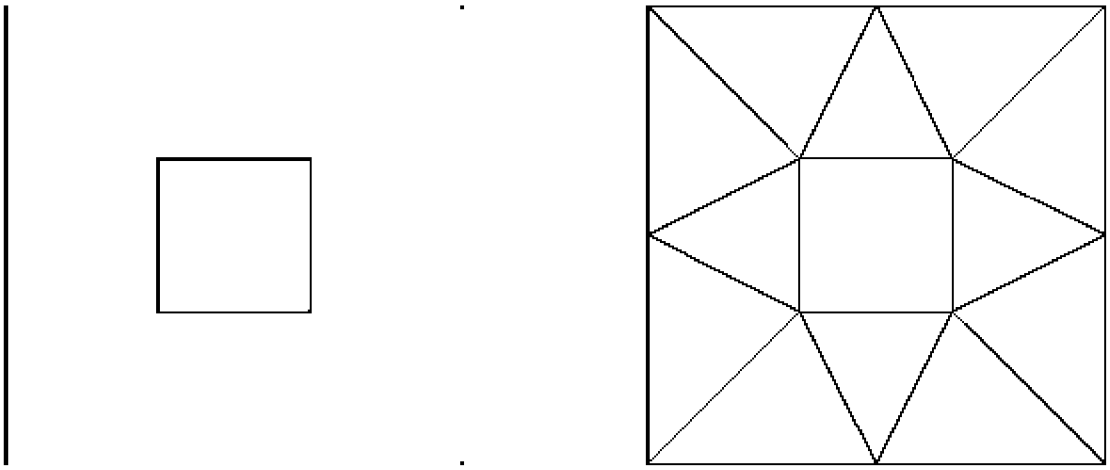
2.1.3 Voronoiov diagram

Tak ako mriežka, aj Voronoiov diagram [12] rozdeľuje priestor na viac menších geometrických obrazcov. Na jeho zostrojenie sú potrebné dva alebo viaceré body z Euklidovskej roviny. Každý z týchto bodov priradíme najbližšiemu bodu v zmysle Euklidovskej vzdialenosti, inému ako je on sám, z tejto Euklidovskej roviny. Výsledkom tohto postupu bude rozdelenie roviny na množinu oblastí asociovaných danej množine bodov. Toto sa nazýva rovinným Voronoiovým diagramom a oblasti ktoré boli vytvorené nazývame Voronoiovými polygónmi.

2.1.4 Delaunayova triangulácia priestoru

Ak je potrebné priestor prehľadávať, je potrebné aby bol systematicky usporiadaný. Na tento účel slúži Delaunayova triangulácia priestoru [12], ktorá preskupí body tak, aby sa s nimi ľahšie manipulovalo. Majme Voronoiov diagram, generovaný konečnou množinou nekolineárnych bodov. Pospájame všetky množiny generátorov, pre ktoré majú Voronoiove polygóny spoločnú hranu. Výsledkom bude rozdelenie roviny – teselácia. Ak táto teselácia pozostáva iba z trojuholníkov, nazývame ju Delaunayova triangulácia, ak nepozostáva z trojuholníkov, nazývame ju Delaunayova pretriangulácia. V Delaunayovej pretriangulácii môžeme rozdeliť netrojuholníkové polygóny

nepretínajúcimi sa úsečkami, ktoré spájajú vrcholy, a tým dostaneme trojuholníky. Výsledkom bude triangulácia, ktorú tiež nazývame Delonayovou triangulaciou.



Obr.3 Vygenerovanie Delaunayovej triangulácie z priestoru tvoreného bodmi a polygónmi [12].

2.2 Metódy popisu stavového priestoru

Metódy vyhľadávania v priestore, na ktoré sa v tejto práci zameriam, kladú dôraz na rýchlosť výpočtu. Toto môžeme ovplyvniť tým, že informácie o priestore, v ktorom bude algoritmus vyhľadávať cestu, reprezentujeme tak, aby to bolo pre algoritmus čo najoptimálnejšie. Reprezentácia priestoru ako stromu je výhodná aj z pohľadu vyhľadávacej zložitosti aj z pohľadu vkladania a mazania prvkov pri tvorení samotného stromu.

2.2.1 K-strom

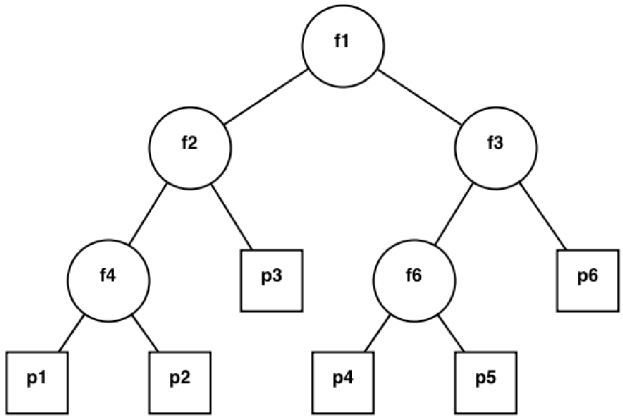
V predchádzajúcej kapitole sme spomenuli karteziánsku sústavu súradníc. Priestor takto reprezentovanej budovy môžeme pomocou algoritmov pre vyhľadávanie cesty v miestnosti s prekážkami, ktoré budú spomenuté neskôr, vytvoriť stavový priestor reprezentujúci budovu ako systém ciest z jednej miestnosti do druhej. Takto vytvorený priestor môže byť popisovaný ako k-strom [3]. Je to priestorovo rozdelená dátová štruktúra pre usporiadanie bodov v k-dimenzionálnom priestore.

K-strom pozostáva z dvoch druhov uzlov. Všetky uzly na vnútorných vetvách, počnúc od koreňa, až po predposlednú úroveň stromu, sú nazývané klastrami. Uzly poslednej úrovne sa nazývajú listami a obsahujú informácie o dátach, ktoré strom uchováva, alebo odkazy na ne. Informácie o zoskupovaní listov, alebo uzlov nižšej vrstvy, obsahujú uzly o úroveň vyššie. Teda informácie o zoskupovaní listov obsahujú uzly predposlednej úrovne stromu.

Algoritmus pre k- strom je zlúčenie b stromu [3] a využitia k-means [3]. Rekurzívna definícia k-stromu je nasledujúca:

1. K_{k+1} je k-strom.
2. Graf, ktorý vznikne z dvoch k-stromov identifikáciou dvoch podgrafov izomorfných K_k, je k strom.

Druhý bod je ekvivalentný s „Graf, ktorý vznikne pripojením vrcholu ku k-stromu tak, že jeho susedstvo indukuje že K_k, je k-strom“ – alebo jeden zo zjednocovaných grafov je práve K_{k+1}.



Obr.4 K-strom – obdĺžniky označujú listy a kružnice uzly stromu.

2.2.2 K-d strom

Modifikovaním klasického binárneho stromu pre viacrozmerné dáta získavame reprezentáciu, ktorú nazývame k-d strom [9]. Je to stromová štruktúra založená na delení priestoru ortogonálnymi nadrovinami. Rieši základný problém určenia poradia v k dimenziách tak, že radí prvky vždy len v jednej dimenzii a na rôznych úrovniach stromu tieto radiace dimenzie pravidelne strieda. Týmto si k-d strom udržuje základné vlastnosti, ako je logaritmický prístup, uchovávanie hodnôt v medziľahlých uzloch a nutnosť vyvažovania stromu pri jeho konštrukcii, kvôli zaručeniu logaritmického času vyhľadávania.

V praxi a tak isto aj v tejto práci, sa stretávame s priestorom ktorý celý poznáme už na začiatku behu aplikácie a netreba ho už neskôr modifikovať. Tým sa vyvažovanie vetiev stromu nahradí vhodným postupom začiatočnej konštrukcie, ktorý rovnomernú výšku všetkých vetiev zabezpečí sám. Vstupom konštrukcie stromu je teda množina k-dimenzionálnych bodov p_1. V našom prípade pri vytváraní aplikácie pre 3D priestor má každý bod 3 súradnice. Najprv sa zvolí k ortogonálnych nadrovín definujúcich orientáciu stromu. V najjednoduchšom prípade sa jedná o nadroviny definované (k-1)-ticami elementárnych vektorov daného priestoru. V prípade nami zvoleného priestoru tak dostaneme 3 plochy definované postupne osami y a z, x a z a nakoniec x a y. Následne sa potom pri zostupe stromom plochy striedajú. Začíname teda v koreni stromu s kompletnou

množinou vstupných bodov. Je ich potreba rozdeliť nadrovinou, rovnobežnou s prvou definovanou základnou nadrovinou. V 3D určíme vzdialenosti bodov od roviny (y,z). Vzdialenosť bodu p_1 od roviny ρ prechádzajúcej počiatkom sústavy súradníc ľahko zistíme pomocou skalárneho súčinu s normálou n_ρ, v tomto prípade osou y:

$$d_i = p_i^t * n_\rho$$

Pokiaľ sa rozhodneme nechať deliace roviny rovnobežné so základnými osami, vykonáva skalárny súčin s jednotkovým vektorom iba výber súradnice. Je teda možné ho vynechať a triediť body priamo podľa jednej z ich súradníc.

Ďalším krokom je vybranie vhodného bodu p_i , ktorý zabezpečí rovnomerné rozloženie množiny bodov. Ideálnym bodom je medián. Spôsob jeho určenia značne ovplyvňuje asymptotickú časovú zložitosť konštrukcie k-dimenzionálneho stromu pre n-bodov:

- Zoradenie všetkých bodov a následne vybranie prostrednej hodnoty v každej úrovni stromu. Konštrukcia celého stromu má zložitosť $O(n \log^2 n)$.
- Zoradenie podľa všetkých k ôs pred samotnou konštrukciou stromu. Zložitosť pre celý strom je v tomto prípade $O(k * n \log n)$ a je vhodná iba ak $k < n$.
- Použitie algoritmu pre nájdenie m-tého prvku postupnosťou radenia. Tu je možné použiť algoritmus radenia mediánu mediánov alebo rekurzívny zostup a tým sa dostať na najhoršiu zložitosť $O(n)$.

Vybraný medián je potom uložený ako uzol stromu a všetky prvky naľavo potom zostupujú rekurzívne do konštrukcie ľavého a pravého podstromu. V každej úrovni je na zostatkovú polovicu bodu aplikovaný rovnaký postup ako v koreni, ale s ďalšou cyklicky menenou deliacou nadrovinou. Algoritmus postupuje až kým v jednej alebo druhej vetve uzlu danej úrovne už nie sú žiadne body. Tu postup zastaví a pokračuje v ďalšej vetve, dokiaľ sú v nej ešte body. Pri dobrom vyvážení, je rozdiel výšok dvoch podstromov najviac jedna.

Postup vyhľadávania v strome je veľmi intuitívny. Postupne od koreňa smerom dole je vždy zistená vzdialenosť bodu od deliacej roviny danej úrovne a porovnaná so vzdialenosťou daného uzla. Ak je hodnota bodu vyššia, postupujeme do pravého podstromu, Ak nižšie, tak do ľavého. Ak je zistená zhoda, je nutné overiť, či sa zhodujú všetky súradnice bodu. V prípade že áno, je vyhľadávanie ukončené, inak zostupujeme do pravého podstromu. Hľadanie končí buď tým, že je nájdená úplná zhoda, alebo algoritmus dôjde do prázdnej vetvy a to znamená, že daný bod v strome nie je.

2.2.3 B-d strom

Ďalšou binárnou stromovou štruktúrou, založenou na delení priestoru, je b-d strom [9]. Je založený na princípe hierarchie obalových, obecne k dimenzionálnych, hypergulí. Konštrukcia je rovnako ako pri k-d stromoch založená na rekurzívnom delení vstupného súboru. Uchováva nielen

body, ale celé primitíva, ako napríklad trojuholníky. Obecne však nemusí ísť o trojuholníky, ale o ľubovoľné elementárne útvary, pri ktorých sme schopný určiť minimálnu, alebo aspoň približne minimálnu obalovú hyperguľu. To znamená prakticky akúkoľvek množinu aspoň dvoch bodov v priestore dimenzie.

Algoritmus na konštrukciu potom potupuje nasledovne [9]:

1. Určí lokálne súradnice množiny vstupného súboru.
2. V smere hlavnej osy rozdelí vstupný súbor geometricky na dve polovice.
3. Každá polovica potom vstupuje do tohto kroku ako vstup do prvého kroku pre konštrukciu ľavého a pravého podstromu.
4. Ak je vo vstupnom bode práve jedna primitíva, táto tvorí list stromu a jeho obalová hyperguľa je obalovou hyperguľou listu.
5. Nasleduje postup znovu hore a v každom poschodí sú budované obalové hypergule vyššej úrovne.
6. Algoritmus končí znovu pri koreni.

Najprv je teda potrebné určiť hlavný smer množiny primitív. Jednou z možností je použiť Principal Component analysis (PCA) [13], ktorá statickou metódou korelácie dokáže určiť smer hlavnej osy. To je taký smer, s ktorým majú body najvyššiu koreláciu. V praxi potom popisujeme najdlhší priebeh množinou. Pri aplikácii na elipsoid algoritmus nájde najdlhšiu os a obecne je jednou z jeho možných aplikácií určenie lokálnych súradníc pre konštrukciu objektovo orientovaných obálok.

Pretože vstupom nie sú body ale celé primitíva P_i, môžeme namiesto nich použiť stredy ich obalových hypergulí. Pomerne jednoduchý a obecný algoritmus na odhad obalovej hypergule spočíva vo výpočte ťažiska vrcholov c_i definovaného ako: $c_i = \frac{1}{|P_i|} * \sum_{p \in P_i} p$ a jeho iteratívnemu posúvaniu k najvzdialenejšiemu bodu po elementárnych krokoch a opätovného prepočítaniu polomeru. Už prvá iterácia a teda samotné ťažisko však dáva rozumnú aproximáciu stredu minimálnej obalovej hypergule. Potom je možné PCA určiť na základe spektrálnej analýzy korelačnej matice M_C ako: $M_C = \frac{1}{n} \sum_{i=1}^{n} c_i * c_i^T$, kde n počet je vrcholov primitíva, c_i sú hodnoty vrcholov a c_i^T sú transponované hodnoty týchto vrcholov v rámci dvoch dimenzií. Samotná analýza spočíva v nájdení vlastných čísel a príslušných vlastných vektorov matice M_c. Vlastné vektory sú na seba ortogonálne a vytvárajú lokálny súradnicový systém vstupného súboru. Vlastný vektor prislúchajúci najväčšiemu číslu je potom hľadanou osou. K rozdeleniu vstupného súboru použijeme geometrický stred získaný ako: $d_c = \frac{\min(di) + \max(di)}{2}$, kde d_c je geometrický stred hodnôt vstupného súboru, $\min(di)$ je najnižšia hodnota vstupného súboru a $\max(di)$ je najvyššia hodnota vstupného súboru. Rozdelením množiny primitív podľa hodnoty vzdialenosti ich stredov, ich rovnobežnou rovinou rozdelíme na dve

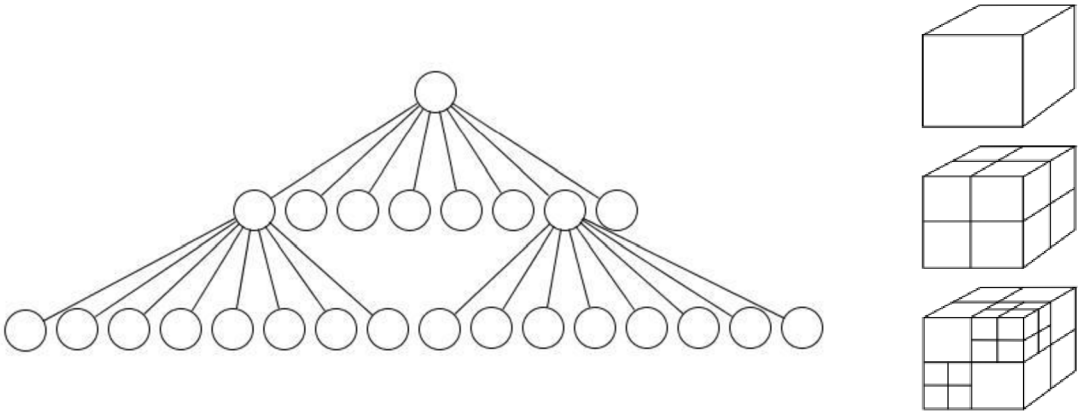
približne rovnako priestorovo veľké oblasti. Je nutné ošetriť výskyt viacnásobných bodov. Táto situácia je ľahko rozpoznateľná tak, že ľavá podmnožina je prázdna. To môže nastať iba ak sú všetky body v pravej množine identické. To sa dá napraviť presunutím bodov do ľavej vetvy. Zostup končí, ak je na vstupe konštrukcie uzlu práve jedno primitívum. Následne je z nej vytvorený list a jeho obalová guľa je použitá ako obalová guľa tohto listu. Potom nasleduje spätný návrat smerom ku koreňu stromu.

2.2.4 Oktálový strom

Na reprezentáciu oktálovým stromom môžeme naraziť v počítačovej grafike, kde je vo veľkej miere využívaná pri modelovaní 3D objektov. Tento druh stromovej reprezentácie je možné využiť aj pri reprezentácii priestoru budovy. V tejto kapitole bude tento druh stromu v krátkosti popísaný.

Oktálový strom nie je nič viac ako dátová štruktúra, ktorá môže byť využitá pre viac vecí. Je užitočná pre detekciu viditeľnosti, ale napríklad aj pre detekciu kolízií. Najdôležitejšie je pochopiť, že pokiaľ rodičovský prvok nie je dôležitý, teda nie je rozdelený na ďalší podstrom, tak nemá potomkov a teda nie je dôležité daného rodiča riešiť, napríklad z dôvodu kolízií.

Definíciu oktálového stromu môžeme popísať nasledovne:
Celú scénu obsahujúcu modelované 3D údaje možno chápať ako kocku. Ak kocka obsahuje rôznorodé dáta, je ďalej rozdelená na ďalších osem kociek.



Obr.5 Model oktálového stromu.

3 Metódy vyhľadávania cesty

Metódy na vyhľadávanie cesty v stavovom priestore môžeme rozdeliť podľa rôznych kritérií. Týmito kritériami môže byť rozmer priestoru v ktorom pracujú, optimálnosť nimi nájdeného riešenia, alebo reprezentácia priestoru s ktorým pracujú. V tejto práci bolo vybrané rozdelenie podľa rozmeru s ktorým pracujú. V nasledujúcich kapitolách budú priblížené vyhľadávacie algoritmy, ktoré sú vhodné na prehľadávanie priestoru budovy s prekážkami.

3.1 Metódy vyhľadávania cesty v priestore s prekážkami

Naša aplikácia pracuje s popisom budovy tj. s priestorom s prekážkami. Preto je potrebné zamerať sa aj na tento fakt a použiť metódu ktorá dokáže prekážky obísť najlepšie ohodnotenou cestou tak, aby dosiahla cieľ.

3.1.1 Lievikový algoritmus (Funnel algorithm)

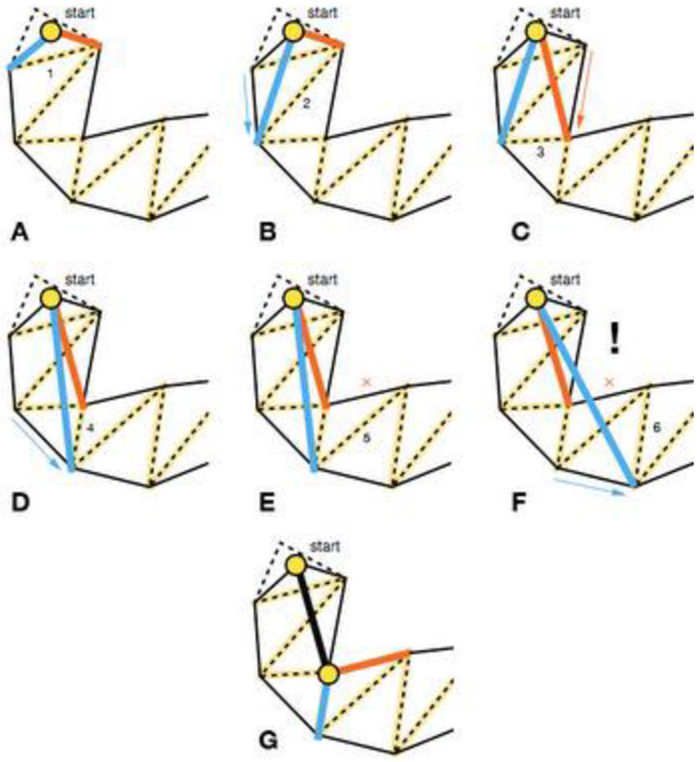
Lievikový algoritmus (z ang. Funnel algorithm) [5] funguje na princípe vyhľadávania najkratšej cesty zo štartovacieho bodu do finálneho, cieľového, bodu v množine trojuholníkov (kanála). Vnútorná hrana je definovaná ako nenútená hrana v triangulácii, ktorou bude objekt prechádzať pri prechode kanálom. Lievikový algoritmus používa nasledovné štruktúry [6]:

1. Cesta – riadkové segmenty najkratšej trasy (s ukladaním bodu ktorý vyberáme).
2. Vrchol – bod ktorý spája cestu a „lievik“.
3. Lievik – dve časti segmentov cesty, z ktorých je jeden v smere hodinových ručičiek a druhý proti smeru hodinových ručičiek, čo predstavuje oblasť v ktorej ešte neboli spracované najkratšie cesty.
4. Klin – tvorí konvexnú reťaz na hornej a dolnej hranici. Je nutné pridať aktuálny vrchol, ktorý sa nachádza pod a nad klinom.

Obrázok 6 ukazuje šesť krokov lievikového algoritmu. Zakaždým, keď ideme spracovať novú hranu (prerušované čiary zvýraznené žltou farbou), skontrolujeme, či sú ľavý a pravý bod vo vnútri už existujúceho lievika. Ak sú, jednoducho lievik predĺžime (A-D). Ak je nový ľavý koncový bod mimo lievika, nie je predĺžený. Ak je nový ľavý koncový bod nad pravým koncovým bodom okraja (F), pridáme pravý bod ako rohový bod v ceste a umiestnime bod namiesto pravého bodu lievika.

Po vykonaní uvedeného postupu reštartujeme algoritmus z takéhoto novozvoleného bodu (G). Rovnaká logika platí aj pre ľavý okraj. Toto sa opakuje až kým nie sú spracované všetky hrany.

Ako je vidieť na vyššie uvedenom príklade, algoritmus prepočíta niektoré hrany viackrát z dôvodu reštartu. Keďže používaný algoritmus je jednoduchý, tento trik robí implementáciu jednoduchšiu a v praxi taktiež rýchlejšiu.



Obr.6 Ukážka práce lievikového algoritmu [5].

3.1.2 Hierarchický A* algoritmus (HPA*)

Existuje mnoho algoritmov na vyhľadávanie cesty a čitateľ by mohol povedať, že už spĺňajú všetky požiadavky, alebo že pre každý problém už existuje algoritmus, ktorý ho rieši najlepšie. Stále sa však objavujú metódy, ktoré sú ideálnejšie pre rôzne situácie. V nasledujúcich kapitolách popíšem metódy, ktoré sú založené na metóde A* (3.2.4) a pridávajú jej vlastnosti, ktoré sú pre určité riešenie úloh v určitých situáciách prínosné.

Metóda HPA* [10] je rozdelená do troch krokov, ktoré nesú vzájomný názov „on-line“ vyhľadávanie. Prvý krok je cestovanie po hranici územia, ktoré obsahuje bod z ktorého štartujeme. Druhým krokom je hľadanie cesty, ktorá spája územie štartovacieho bodu a územie cieľového bodu. Nakoniec je potrebné nájsť cestu z hraničnej oblasti cieľovej oblasti k cieľovému bodu. Abstraktný graf pre on-line vyhľadávanie je vytvorený pomocou informácií, ktoré poznáme o priestore v ktorom máme vyhľadávanie uskutočniť.

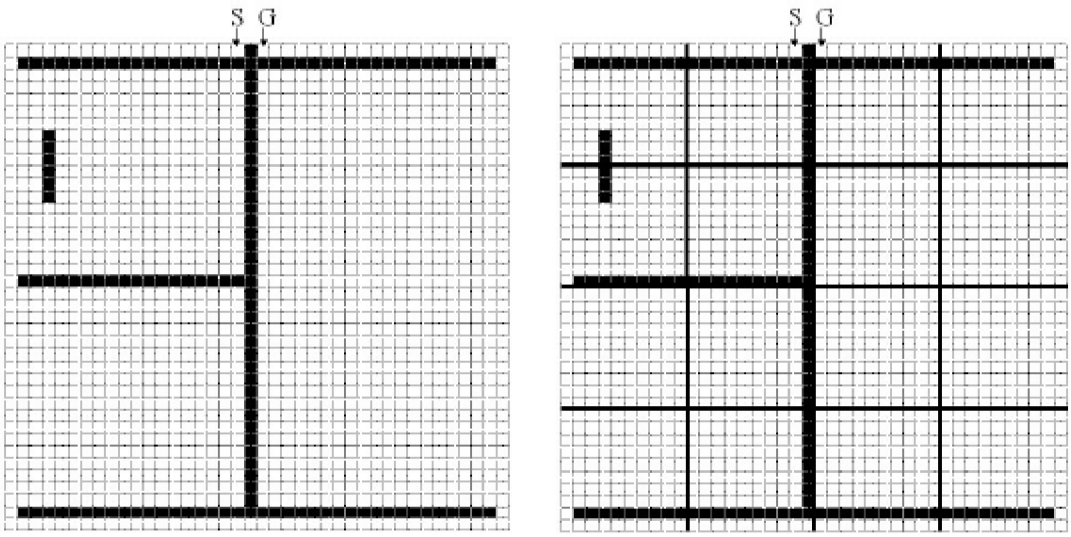
Graf je vytváraný v dvoch úrovniach, tj. hlavná, detailná a abstraktná úroveň.

Prvým krokom vo vytváraní štruktúry pre hierarchické vyhľadávanie je definovanie topologickej abstrakcie reálneho sveta (bludiska, budovy). Táto abstrakcia je tvorená množinou disjunktných štvorstenov, ktoré majú pomenovanie „klaster“. Pre každú hraničnú čiaru medzi dvomi klastrami identifikujeme množinu prechodov medzi nimi. Prechod je maximálne hraničné miesto dvoch klastrov c_1 a c_2, v ktorom nie sú žiadne prekážky, formálne definované nasledovne:

Zoberme dve susedné línie dlaždíc l_1 a l_2, jednu z každého klastra, ktoré určujú hranicu medzi týmito klastrami. Pre dlaždice $t \in l_1 \cup l_2$. definujeme symetrickú dlaždicu t ktorá slúži ako hranica medzi c_1 a c_2.

Prechod e je definovaný ako množina dlaždíc ktoré dodržujú nasledovné pravidlá:

- $e \in l_1 \cup l_2$, to znamená že prechod je definovaný pozdĺž a nemôže prekročiť hranicu medzi dvomi priľahlými klastrami.
- Podmienka symetrie: $\forall t \in l_1 \cup l_2 : t \in e$.
- Bezbariérová podmienka: Prechod medzi klastrami neobsahuje žiadne prekážky.



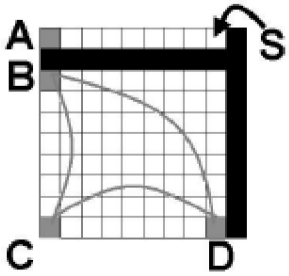
Obr.7 Ukážka rozdelenia priestoru na klastre [10].

Prechody sa používajú na vytvorenie abstraktného grafu. Pre každý prechod definujeme dva uzly v abstraktnom grafe a hranu medzi nimi ktorá ich spája. Vzhľadom k tomu, že hrana reprezentuje prechod medzi dvoma klastrami, nazývame ju vonkajšia hrana. Tieto hrany majú vždy dĺžku jedna. Pre každú dvojicu uzlov vo vnútri klastra definujeme hranu ktorá ich prepája a nazývame ju vnútorná hrana. Dĺžku vnútorných hrán počítame ako optimálnu cestu vo vnútornej oblasti klastra medzi uzlami ktoré spája táto hrana pomocou A* algoritmu, ktorý bude popísaný v nasledujúcej kapitole (3.2.4), s heuristickou funkciou definovanou ako Euklidova vzdialenosť týchto bodov. Keď je abstraktný graf vytvorený pre všetky klastre, je pripravený na prehľadávanie. Toto je výhodné pre prehľadávanie priestoru ktorý vopred dobre poznáme.

Prehľadávanie začína pridaním štartovacieho a cieľového bodu do abstraktného grafu. Potom ako sú body vložené, využijeme klasický A* algoritmus na vyhľadanie cesty medzi štartovacím bodom a cieľovým bodom. To znamená skutočné presunutie od štartovacieho bodu z hranice jeho klastru, presunutie k hranici klastru cieľového bodu abstraktným grafom a následné presunutie od hranice klastru k cieľovému bodu.

Po ukončení algoritmu môže byť abstraktná cesta zjemnená na získanie detailnejšej reprezentácie cesty zo štartovacieho do cieľového bodu. Aby to bolo možné, musia byť štartovací a cieľový bod časťami grafu. Pre oba body je algoritmus zjemňovania rovnaký, preto ho v tejto práci predstavím iba na štartovacom bode.

Spojíme štartovací bod s hranicou jeho klastru. Nájdeme optimálnu cestu medzi daným bodom a všetkými ostatými abstraktnými bodmi daného klastru. Ak taká cesta existuje, pridáme ju do abstraktného grafu a stanovíme jej dĺžku dĺžkou cesty. Lokálny cieľový a štartovací bod sa menia pre každé nové vyhľadávanie. Po nájdení cesty sa cieľový a koncový bod odstránia z grafu. Nakoniec sa každý prechod jednotlivým klastrom nahradí ekvivalentom reálnych prechodov.



	A	B	C	D
A		∞	∞	∞
B			7.00	10.94
C				8.00
D				

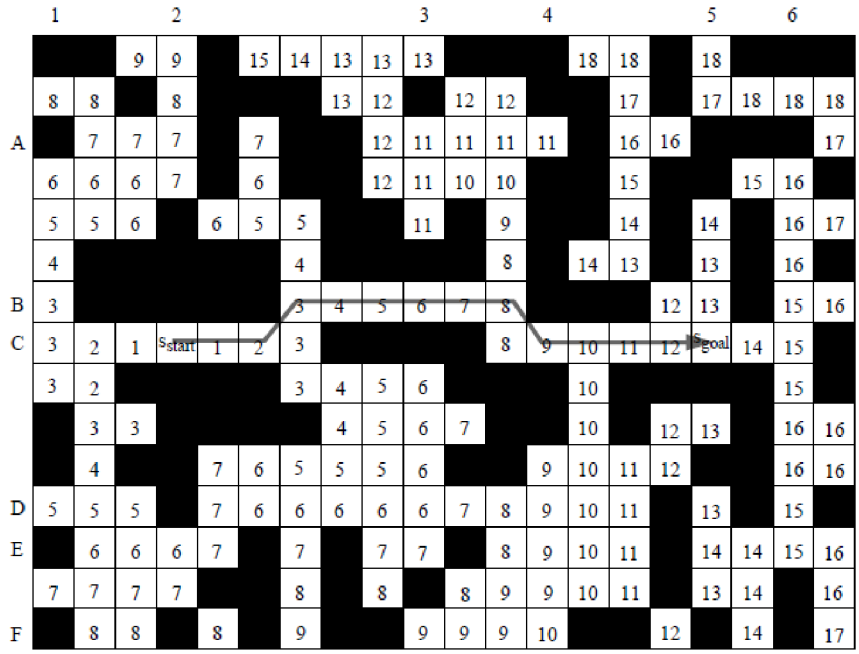
Obr.8 Ukážka vnútorného prepojenia uzlov jedného z klastrov [10].

HPA* algoritmus je možné rozšíriť do viacvrstvovej hierarchie a je nezávislý na pôvodnej mape. Jeho odchýlka od optimálnej cesty je relatívne malá iba 1%, čo je vo väčšine prípadov prijateľný výsledok. Pri skupinovom vyhľadávaní ciest sa vyskytuje problém pri ktorom všetky jednotky z toho istého klastru dostanú v podstate rovnaké cesty.

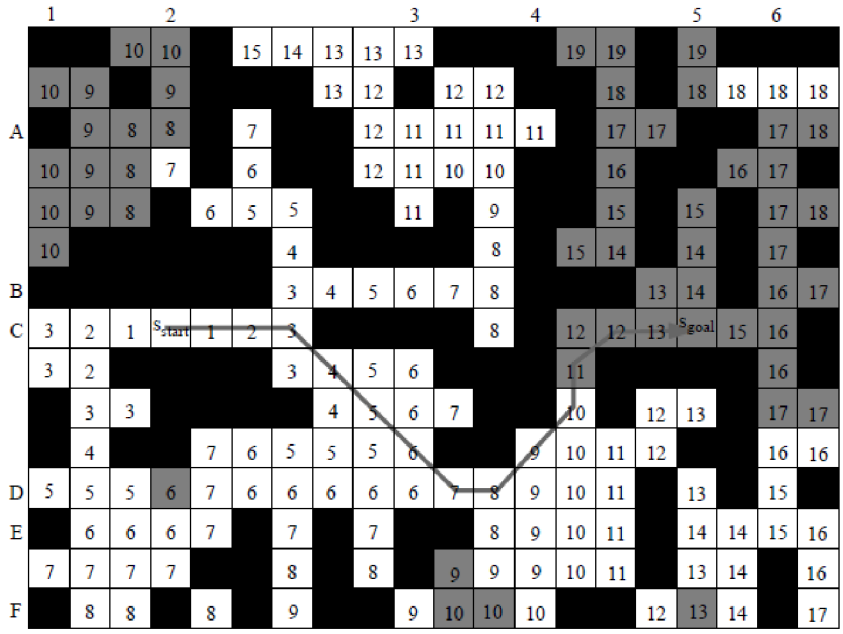
3.1.3 Doživotné A* (LPA*)

Ďalšou metódou na vyhľadávanie cesty v priestore ktorá vylepšuje A* (3.2.4) je LPA* [11]. Tento algoritmus je výhodné použiť pri vyhľadávaní cesty v priestore, ktorý sa môže meniť v závislosti na čase. Všetkým dlaždiciam v priestore priradíme hodnotu, ktorá vyjadruje dĺžku cesty z cieľového bodu do všetkých uvažovaných dlaždíc. V počiatočnej fáze, keď je známy priestor ktorý ideme prehľadávať, je nájdenie optimálnej cesty zo štartovacieho do cieľového bodu ľahké a vykonáva sa pomocou A* algoritmu s heuristickou funkciou definovanou ako Euklidova vzdialenosť dvoch bodov . Po zmene priestoru potrebujeme zistiť pozíciu, kde sa nachádza bod,

v ktorom algoritmus skončil. Táto pozícia je vyznačená v prvotnom aj v pozmenenom priestore. Keďže niektoré ohodnotenia dlaždíc neboli pozmenené, nemusia byť prepočítavané.



Obr.9 Model zobrazujúci ohodnotený počiatočný priestor a vyhľadanú cestu [11].



Obr.10 Model zobrazujúci pozmenený priestor s prepočítanými ohodnoteniami a vyhľadanú cestu. Pozmenené a prepočítavané uzly sú zafarbené šedou farbou [11].

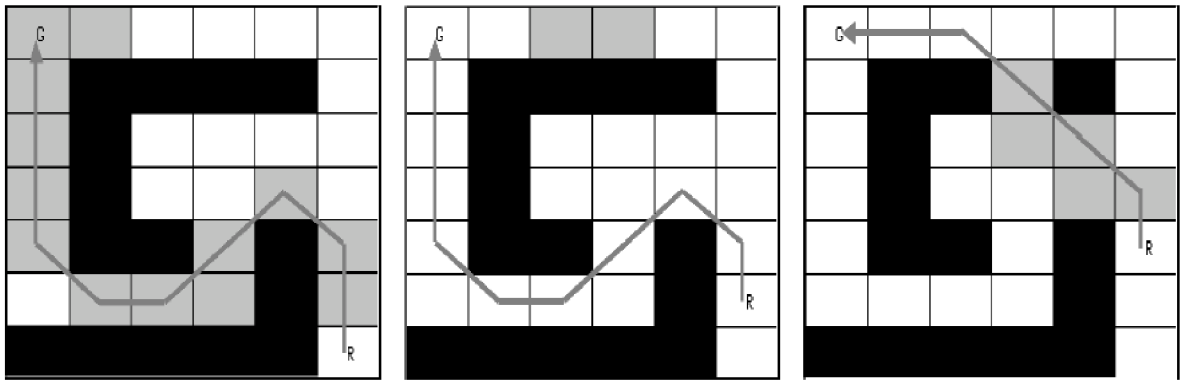
3.1.4 Kedykoľvek Dynamický A* (Anytime Dynamic A*)

V tejto časti priblížim kedykoľvek dynamický A* (z ang. Anytime Dynamic A*) [7], ktorý je založený na A* algoritme (3.2.4). Taktiež je výhodný pri vyhľadávaní cesty v dostupnom čase, kde pri každom kroku znovu používa už vypočítané výsledky a priestor, ktorý algoritmus prehľadáva,

nemusí byť dopredu známy. Akonáhle sú zistené nové údaje o grafe, algoritmus postupne opraví pôvodnú cestu tak, aby vyhovovala zisteným faktom. Výsledkom je postup, ktorý poskytuje efektívne riešenie zložitých a dynamických problémov.

Ako bolo uvedené vyššie, často sa stáva, že informácie o priestore sú nedokonalé a neúplné. V dôsledku toho akékoľvek riešenie generované v počiatočnej fáze môže byť neplatné, alebo neoptimálne, keď sa získajú ďalšie dáta o prehľadávanom priestore. Je preto dôležité, aby bol algoritmus schopný pozmeniť pôvodne naplánovanú optimálnu cestu pri zistení nových údajov. Niekedy je však čas na nájdenie trasy obmedzený a preto sa uspokojíme aj s menej optimálnou cestou. Užitočné algoritmy pre generovanie takýchto riešení sa nazývajú „anytime algoritmy". Typicky sa jedná o nájdenie cesty pomocou počiatočných znalostí a ak to čas dovolí, je táto cesta optimalizovaná.

Kedykoľvek dynamický A* tak ako A* používa fronty CLOSE a OPEN v ktorých uchováva už prejdené a ešte dostupné uzly. Ak dôjde k zmenám, ktoré ovplyvnia hodnotu uzlov v grafe prostredia, ovplyvnené uzly sú vložené do fronty ešte neprejdených uzlov s takou prioritou, ktorá sa rovná ich minimálnej hodnote pred zmenou a hodnote ich kľúča po zmene. Následne sú uzly znovu prehľadávané kým nie je nájdená cesta. Ak je dosiahnutie cieľového bodu uskutočnené do potrebného času, ako cesta sa požije nová cesta, v opačnom prípade sa použije stará cesta a znovu sa začína optimalizácia tejto cesty z nového štartovacieho bodu po jeho posunutí. Ako heuristická funkcia je použitá Euklidova vzdialenosť medzi cieľovým a aktuálnym bodom.



Obr.11 Hľadanie cesty v jednoduchom priestore s Kedykoľvek Dynamický A*. Začína sa v pravom dolnom rohu, z ktorého je nájdená cesta do ľavého horného rohu. Po prejdení dvoch krokov je objavená zmena v priestore. Po zaznamenaní tejto skutočnosti je zahájené nové vyhľadávanie cesty ktorou bude algoritmus následne postupovať [7].

3.2 Metódy vyhľadávania cesty v 2D stavovom priestore

Pri prehľadávaní stavového priestoru je potrebné zvoliť si priority, podľa ktorých budeme vyberať vyhovujúci algoritmus. Pri hľadaní najkratšej cesty je teda dôležité, aby bol algoritmus vyhľadávania cesty optimálny. Taktiež môžeme vziať v úvahu, že graf, v ktorom budeme cestu vyhľadávať, bude vždy kladne ohodnotený. Nasledujúce kapitoly popíšu rôzne vyhľadávacie algoritmy v stavovom priestore a priblíži ich základný princíp.

3.2.1 Vyhľadávanie do šírky (BFS)

Vyhľadávacie algoritmy môžeme rozdeliť na dve skupiny. Neinformované metódy, ako je napríklad BFS [2], ktorý v tejto kapitole rozoberiem podrobnejšie a informované metódy, kde patrí napríklad Dijkstrov algoritmus, ktorý bude popísaný neskôr (3.2.2).

Neinformované metódy vyhľadávania nemajú k dispozícii žiadne informácie o nasledovníkoch uzla, alebo o stavovom priestore, ktoré by im pomohli urýchliť proces vyhľadávania.

BFS je jeden z najjednoduchších algoritmov na vyhľadávanie a slúži ako základ pre mnoho dôležitých vyhľadávacích algoritmov.

Majme graf $G = (V, E)$ a vrchol grafu s. BFS systematicky prehľadáva hrany grafu G na objavenie každého vrcholu, ktorý je dosiahnuteľný z s. Počíta vzdialenosť (najmenšie ohodnotenie hrán) z s do každého dosiahnuteľného uzla. Taktiež produkuje stromovú štruktúru s koreňom, ktorá obsahuje všetky dosiahnuteľné uzly. Pre každú hranu dosiahnuteľnú z s platí, že cesta v štruktúre odpovedá najkratšej ceste z s do v v G.

Algoritmus môže pracovať s orientovanými aj neorientovanými grafmi. BFS sa volá „vyhľadávanie do šírky“ preto, lebo rozširuje hranice medzi objavenými a neobjavenými uzlami rovnomerne po celej šírke grafu. To znamená, že algoritmus najprv objaví všetky uzly vo vzdialenosti od k do s predtým, ako objaví nejaké vo vzdialenosti $k+1$.

Pre udržanie prehľadnosti, BFS „farbí“ každý uzol na bielu, šedú alebo čiernu farbu. Všetky vrcholy začínajú ako biele. Ak pri vyhľadávaní narazíme na vrchol ktorý je biely, vieme že vrchol ešte nebol prehľadávaný. Pokiaľ $(u, v) \in E$ a uzol u je čierny, potom uzol v je buď šedý alebo čierny. To znamená, že všetky uzly susediace s čiernym uzlom už boli objavené. Šedé uzly môžu mať niektoré uzly objavene a niektoré neobjavené, tvoria hranicu medzi nimi.

Ako už bolo spomínané, BFS vytvára stromovú štruktúru, spočiatku obsahujúcu iba jeden koreň, ktorý je štartovací bod. Akonáhle je objavený biely uzol v, ako následník už objaveného uzla u, je pridaný do tejto štruktúry. Hovoríme že uzol u je predchodca, alebo rodič uzla v. Od objavenia danného uzlu, tento uzol môže mať maximálne jedného rodiča.

3.2.2 Dijkstrov algoritmus

Jeden zo základných algoritmov na prehľadávanie stavového priestoru je Dijkstrov algoritmus [2]. Tvorí základ pre množstvo vyhľadávacích algoritmov ako je napríklad A*, ktorý predstavím neskôr. V tejto kapitole popíšem už zmieňovaný Dijkstrov algoritmus.

Dijkstrov algoritmus môže byť jednoducho pomenovaný ako váhovo ohodnotené vyhľadávanie do šírky. Označme pre každý uzol $v \in V$ aktuálnu nezáväznú vzdialenosť od počiatočného uzla ako $d(v)$ a jeho predchodca na najkratšej ceste ako $p(v)$. Navyše každý uzol dostane stav $S(v)$, ktorý môže nadobúdať hodnoty (nenavštívený, aktuálny, navštívený). Pri inicializácii majú všetky uzly stav nenavštívený, vzdialenosť $d(v) = 1$ a predchodca $\mathrm{p}(v) = null$. Tieto uzly, ktoré sú v súčasnej dobe označované podľa algoritmu, sú organizované v prioritnej fronte Q. V inicializácii je zdrojový uzol označený s a je mu pridelená hodnota $d(s) = 0$. Pokiaľ nie je prioritná fronta prázdna, algoritmus vyberá prvok v z fronty Q s najmenšou aktuálnou vzdialenosťou, skontroluje a nastaví jeho status na navštívený. Pojem „kontrola uzla“ znamená, že všetky odchádzajúce hrany $(v, u) \in E$ sú skontrolované a ak cesta $s - \cdots - v - u$ je kratšia ako $d(u)$, vzdialenosť $d(u)$ bude aktualizovaná ako $d(v) + l(u, v)$ a predchodca je nastavený na $p(u) = v$. Ak je uzol u nenavštívený, je vložený do priority a jeho nový status je aktuálny. Ak je jeho status už aktuálny, je aktualizovaná jeho priorita (znížená).

Jeho najhoršia časová zložitosť záleží na type prioritnej fronty.

3.2.3 UCS – Uniform cost search

UCS je algoritmus ktorý prehľadáva stromové štruktúry a je používaný pre vyhľadávanie vo vyváženom strome, stromovej štruktúre alebo grafe.

Hľadanie začína pri koreňovom uzle. Následne pokračuje na nasledujúcom uzle, ktorý má najmenšie náklady na cestu z koreňa. Algoritmus pokračuje až kým nie je nájdený cieľový uzol.

Algoritmus zahŕňa pridanie hodnoty všetkých neexpandovaných uzlov, ktoré sú prepojené s priamou cestou k prioritnej fronte expandujúcim uzlom. Vo fronte je každý uzol spojený úplnou hodnotou zo štartovacieho uzla, kde najväčšia priorita je daná bodu s najmenším ohodnotením. Uzol ktorý je na vrchu fronty, je následne vyhodnotený pridaním ďalšej sady bodov, ktoré sú k nemu susediace, s celkovými nákladmi od koreňa k príslušnému uzlu.

Tento algoritmus je úplný, ak je počet nasledovníkov každého uzlu vždy konečný, čo je pre náš problém vždy pravdivé.

3.2.4 A* algoritmus

Algoritmus A* [4] je používaný pre vyhľadávanie optimálnych ciest v kladne ohodnotených grafoch. Používa obdobný algoritmus ako Dijkstrov algoritmus a pridáva k nemu heuristiku.

Používa princíp hladovania z daného počiatočného bodu do požadovaného cieľového bodu. Optimálnou cestou sa v tomto algoritme rozumie cesta s najlepším ohodnotením. Cesta je v závislosti od reprezentácie hodnôt cien hrán.

A* používa funkciu f(x), ktorá ohodnocuje jednotlivé uzly pre určenie postupnosti spracovania týchto uzlov. Táto funkcia je tvorená dvomi ďalšími funkciami g(x) a h(x) kde g(x) predstavuje cenu prekonania vzdialenosti medzi začiatočným uzlom a uzlom aktuálnym a h(x) predstavuje heuristickú funkciu. Táto heuristická funkcia odhaduje optimálnosť postupu cesty za pomoci ceny prekonania vzdialenosti z aktuálneho uzlu do cieľového uzlu. Zároveň musí byť prípustná tzn. nemôže nadhodnocovať vzdialenosť k cieľu. V našom prípade bude použitá ako heuristická funkcia vzdialenosť vzdušnou čiarou, keďže je to fyzicky najkratšia cesta. V prípade, že heuristická funkcia spĺňa podmienku:

$$h(x) \leq d(x, y) + h(y)$$

pre každú hranu x, y grafu (c je dĺžka tejto hrany), potom je h monotónna. V tomto prípade to znamená že navštívi každý uzol maximálne jedenkrát.

Algoritmus prebieha nasledovne: Je vytvorená a udržiavaná prioritná fronta otvorených, tj. ešte nespracovaných uzlov. Čím je menšia hodnota f(x) pre uzol x, tým vyššiu prioritu uzol má. V každom kroku algoritmu je uzol s najvyššou prioritou odobraný z prioritnej fronty a sú vypočítané hodnoty f(x) a h(x) pre s ním susediace uzly. Tieto uzly sú potom pridané do prioritnej fronty. Algoritmus pokračuje až kým nemá konečný uzol menšiu hodnotu f(x) ako ľubovoľný uzol v prioritnej fronte, alebo dovtedy, kým táto fronta nie je prázdna. Hodnota koncového uzlu je potom dĺžkou najkratšej cesty grafom zo štartovacieho do cieľového bodu grafu. V prípade, že je potrebné poznať aj konkrétnu cestu, je nutné si uchovávať zoznam uzlov na tejto ceste.

3.3 Metódy vyhľadávania cesty v 3D priestore

Vyhľadávanie v 3D priestore pre potreby mnou navrhnutej aplikácie môžeme poňať ako vyhľadávanie v 2D priestore s prechodmi medzi poschodiami. Toto je možné, pretože hľadaná cesta je určená pre pohyb po podlahe budovy, teda v 2D priestore, pri ktorom sa objekt potrebuje dostať z miestnosti A do miestnosti B. Preto je z-ová súradnica súradnicového systému prirodzené číslo, kde toto číslo reprezentuje poschodie.

Prechody medzi poschodiami môžeme realizovať ako prechody medzi dvoma miestnosťami. Rozdiel medzi prechodom v rámci jedného poschodia a prechodom medzi dvoma poschodiami je nasledujúci: Prechod v rámci jedného poschodia je ohodnotený nulovou hodnotou, keďže aj cena prekonania prechodu je nulová. Prechod medzi poschodiami je časovo náročnejší, teda aj cena prekonania týchto prechodov je väčšia v porovnaním s dverami. V navrhnutej aplikácii sú dva prípady viacposchodových prechodov, ktoré sa môžu v priestore nachádzať a tými sú výťah a schody.

3.3.1 Postup vyhľadávania cesty v budove

Postup vyhľadávanie môžeme rozdeliť na viac logických celkov. Tieto celky už boli rozoberané v predchádzajúcich kapitolách, preto tu popíšem iba postup ich spolupráce a zlúčenie do jednej aplikácie.

1. Budovu ktorá je reprezentovaná v karteziánskej sústave súradníc pretransformujeme do stavového priestoru tak, že pre každú miestnosť vyhľadáme cestu do priľahlých miestností tak, aby tieto cesty boli čo najkratšie a obchádzali prekážky ktoré, sú v miestnosti umiestnené. Pre miestnosť, v ktorej je umiestnený štartovací bod a pre miestnosť, v ktorej je umiestnený cieľový bod, sa vyhľadá cesta z tohto bodu do všetkých priľahlých miestností.
2. Do takto vytvoreného stavového priestoru je následne potrebné pridať prechody medzi poschodiami. Vyhľadáme párové prechody, ktoré sú umiestnené medzi poschodiami, ohodnotíme ich podľa typu prechodu a vytvoríme spojenie medzi miestnosťami rôznych poschodí ktoré tento prechod spája.
3. V stavovom priestore ktorý nám reprezentuje viacposchodovú budovu je ako posledný krok na nájdenie cesty vyhľadávanie. Začína sa v miestnosti so štartovacím bodom a pokračuje pokiaľ nie je dosiahnutá miestnosť s cieľovým bodom.

3.3.2 Algoritmus „Od dvier k dverám“ (Door-to-door)

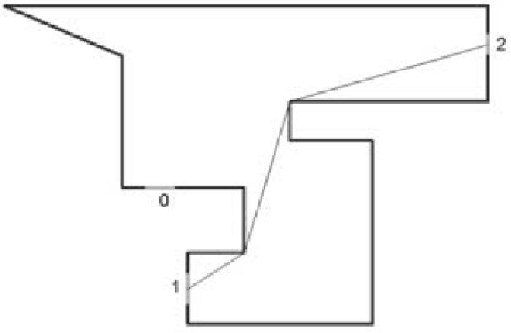
Vyhľadávanie cesty v budove je v našom prípade špecifikované pre peší pohyb z miestnosti v ktorej sa na začiatku nachádzame do miestnosti do ktorej sa chceme dostať.

Algoritmus „od dvier k dverám“ [8] je interpretovaný ako priama pešia cesta z dverí do ďalších viditeľných dverí, alebo najkratšia cesta medzi dvomi neviditeľnými dverami, tzn. ich viditeľnosti bráni prekážka. To však neznamená, že ľudia musia striktne dodržiavať túto cestu. Algoritmus stanovuje potencionálne efektívne cesty pre vnútornú navigáciu. Tento algoritmus je založený na nasledovných predpokladoch:

- Sémanticko-geometrický model budovy je známy a je rozhodujúci pre automatizáciu vyhľadávania cesty, vzhľadom k tomu, že poskytuje umiestnenie, typ a stav dverí (tj. odomknuté alebo zamknuté).
- Model poskytuje priamo alebo nepriamo informácie o pripojení miestností.
- Model obsahuje informácie o všetkých vnútorných objektoch, ktoré sa môžu stať prekážkou v určitom okamžiku.
- Dynamické zmeny v interiéri a štruktúra budovy sú tiež známe, aby na ne mohol algoritmus reagovať prepočítaním hľadanej trasy.

Algoritmus prezentovaný nižšie je typický stromovo zameraný algoritmus, avšak v našom prípade sú dvere alebo iné otvory uzlami a izby sú hranami. To je od väčšiny dostupných algoritmov rozdiel, lebo tieto algoritmy popisujú dvere ako hrany spájajúce miestnosti tj. uzly.

Nasledujúci obrázok ilustruje najkratšiu trasu od dverí 1 k dverám 2 v konkávnej miestnosti. Dva konkávne rohy sú prijaté ako medziľahlé miesta trasy.



Obr.12 Model algoritmu „od dvier k dverám“ pre nájdenie najkratšej cesty v konkávnej miestnosti [8].

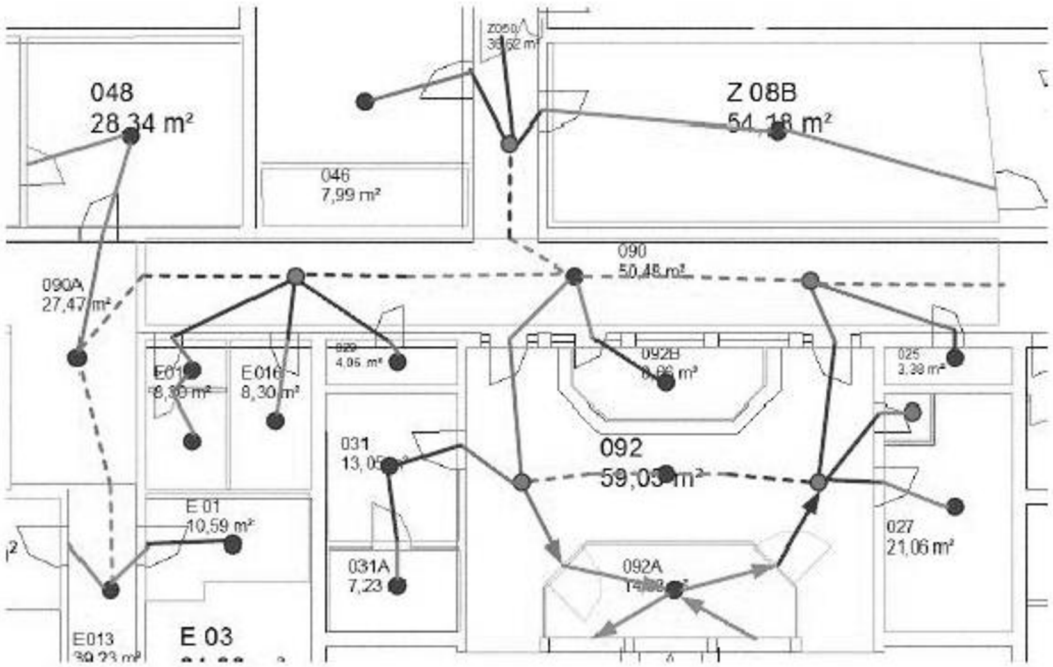
Tento odstavec pojednáva o smerovaní medzi izbami a dá odpoveď na otázku „ako sa pohybovať z miesta na miesto v jednom poschodí“. Tu nás zaujímajú miestnosti, ktorými je možné prejsť. Na základe sémantiky a geometrie jednoduchej budovy môže byť definovaná konektivita miestností. Pre všetky miestnosti treba špecifikovať počet vedľajších miestností, ktoré sú k nej pripojené pomocou dverí. Následne môže byť zostrojený klasický graf, ktorého uzly predstavujú izby a hrany naznačujú spojenie s ďalšími miestnosťami. Najskôr je špecifikovaná štartovacia a cieľová miestnosť. Následne je potrebné vybrať algoritmus, ktorý bude vyberať, ktorá miestnosť bude ďalšia prechádzaná. V metóde „od dvier k dverám“ je použitý A* algoritmus. Nákladmi na prejdenie cesty zvolíme čas potrebný na jej prekonanie, alebo dĺžku tejto trasy.

Tento odstavec diskutuje o smerovaní v jednej miestnosti. Po zadaní štartovacieho a cieľového bodu na jednom poschodí budovy a potom, čo je zistené, ktoré miestnosti sú prístupné, môže smerovanie v miestnosti začať. Keď aktuálny priebeh cesty dôjde ku dverám miestnosti, ktoré sú pripojené k ďalšej miestnosti, je trasa pre navigáciu v ďalšej miestnosti nájdená. Existuje niekoľko krokov ako získať túto trasu:

1. Rozpoznať štartovacie a cieľové dvere. Potom detekovať viditeľnosť medzi nimi. Ak existuje viac dverí do ďalšej miestnosti, vybrať tie, ku ktorým vedie najkratšia cesta.
2. V prípade, že sú tieto dvere viditeľné, prejsť do ďalšej miestnosti.
3. Zistiť konkávne body v miestnosti (polygón v 2D pôdoryse).
4. Zostrojiť graf obsahujúci štartovacie dvere, cieľové dvere a všetky konkávne body v tejto miestnosti. Vytvoriť maticu susednosti v grafe.
5. Pre každý uzol je potrebné určiť viditeľné uzly z neho a vypočítať ich vzdialenosti ako váhu do matice susednosti.

6. Pre cestu do ďalšej miestnosti sa použije najkratšia zistená cesta grafom zo štartovacích do cieľových dverí, na ktorej výpočet je použitý algoritmus A*(3.2.4) alebo HPA*(3.1.2).

Po týchto krokoch získame cestu zo štartovacej miestnosti do cieľovej miestnosti v rámci jedného poschodia. Algoritmus si vyberá nasledovné miestnosti podľa poradia definovaného v predchádzajúcom odstavci. Staré cieľové dvere sa stanú novými štartovacími, nové cieľové dvere sú identifikované a algoritmus znovu začína od prvého kroku. Táto postupnosť sa opakuje pokiaľ nieje aktuálna štartovacia miestnosť zároveň aj cieľovou.



Obr.13 Model algoritmu „od dvier k dverám“ v budove [8].

4 Implementácia

Táto kapitola sa zaoberá vlastnou implementáciou aplikácie. Na začiatku bolo nutné navrhnúť objektovú reprezentáciu budovy, ktorá bude serializovaná pre potreby uloženia mapy budovy s ktorou sa bude pracovať. Následne bolo potrebné túto reprezentáciu vizualizovať, aby bolo možné zobraziť vyhľadanú cestu. V poslednej časti práce bolo potrebné implementovať vhodne vybrané algoritmy na prehľadávanie 3D priestoru, ktoré vyhovovali požiadavkám na našu aplikáciu.

Implementácia prebiehala v jazyku C# v prostredí Microsoft Visual Studio 2012. Keďže vyhľadávanie cesty v zložitejšej budove je časovo viac náročné, tento výpočet sa vykonáva v samostatnom vlákne a práca s aplikáciou, pre užívateľa počas toho výpočtu, nie je blokovaná.

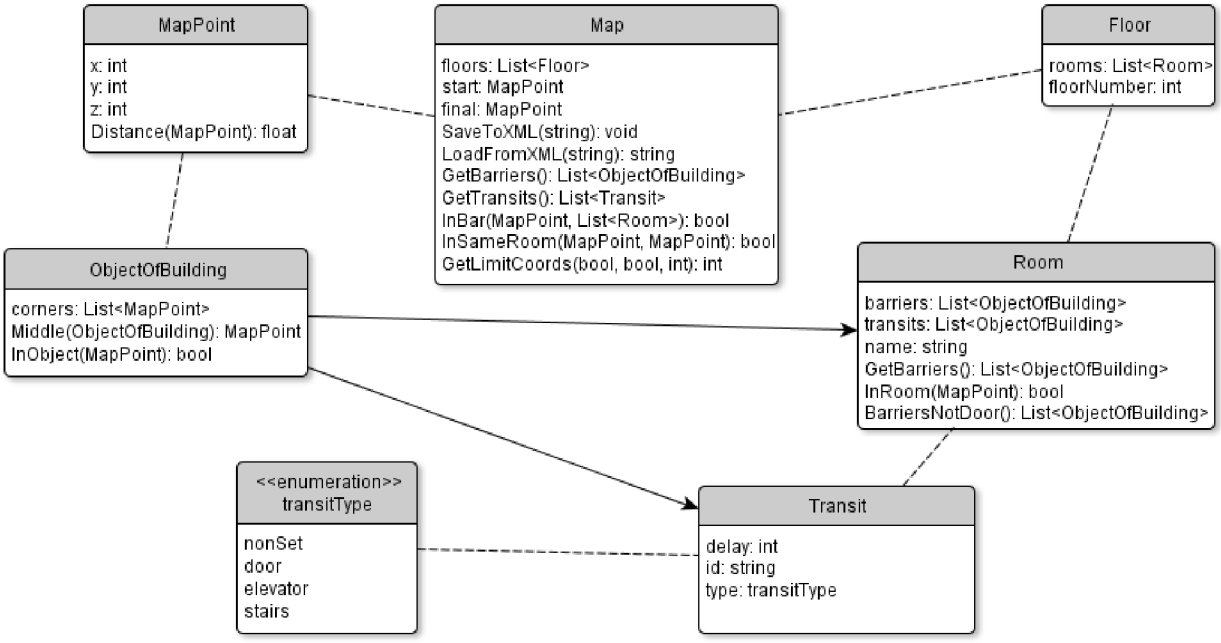
Implementácia aplikácie bola rozdelená do menších celkov a pre každý z týchto celkov bola implementovaná knižnica tried, ktorá ho rieši. Týmito knižnicami tried sú:

1. MapObjectTypes – obsahuje implementáciu objektov budovy.
2. DrawTypes – metódy pre vizualizáciu budovy.
3. AStarTypes – metódy vyhľadávania cesty pomocou A*.
4. DelaunayTriangulationTypes – metódy zostrojenia Delaunayovej triangulácie.
5. HPATypes – metódy vyhľadávania cesty pomocou HPA*.
6. PathfindingApp – zlučuje knižnice do použiteľnej aplikácie.

4.1 Reprezentácia budovy

Budova je reprezentovaná objektmi odvodenými z reálneho sveta, ktoré sa nachádzajú v knižnici tried MapObjectTypes. Základná stavebná jednotka je trojrozmerný bod karteziánskeho súradnicového systému. Tento je definovaný ako trieda MapPoint. Druhým základným typom je ObjectOfBuilding. Je to rodičovská trieda, z ktorej dedia všetky ostatné objekty. Obsahuje množinu hraničných bodov typu MapPoint, ktoré definujú hranice objektu budovy. Ďalšou triedou je Room ktorá reprezentuje miestnosť budovy. Táto obsahuje množinu prekážok v miestnosti typu ObjectOfBuilding, množinu prechodov typu Transit a názov ktorý ju popisuje. Medzi miestnosťami budovy môžu byť tri druhy prekážok a tými sú dvere, schody a výťah. Každý z nich má jednotnú identifikáciu aby bolo možné nájsť párové prechody a hodnotu ktorá určuje zdržanie pri prekonávaní daného prechodu. Miestnosti sú zoskupené do poschodí v triede Floor, ktorá obsahuje túto množinu miestností a jednoznačné číslo poschodia. Všetky poschodia sú uložené ako množina v objekte typu Map, ktorý okrem nich obsahuje ešte reprezentáciu štartovacieho a cieľového bodu, ktoré sú typu MapPoint.

Ukladanie a načítavanie mapy budovy je implementované pomocou serializácie a deserializácie týchto objektov do XML formátu.



Obr.14 Class diagram knižnice tried MapObjectTypes.

4.1.1 XML pre popis budovy

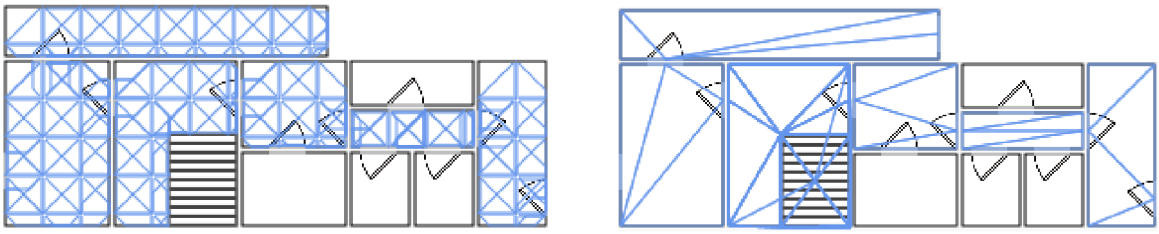
Keďže implementovaná aplikácia neimplementuje editor máp, je nutné popísať štruktúru XML súboru, ktorý uchováva informácie o mape. Na túto úlohu slúži definičný jazyk DTD (Document Type Definition). Súbor s definíciou štruktúry pomocou DTD je priložený k práci ako príloha 1.

4.2 Vizualizácia

Aplikácia je typu Windows Presentation Foundation (WPF) [14]. Toto je zvolené z dôvodu ľahšej manipulácie s prezenčnou vrstvou aplikácie. Hlavná ponuka aplikácie sa nachádza v hornom menu. Tu je možné otvárať a ukladať súbory mapy budovy, vyberať prehľadávací algoritmus, spúšťať a pozastavovať vyhľadávanie cesty, vkladať štartovací a cieľový bod a nachádza sa tu ponuka nápovedy. Pri štarte aplikácie sa otvorí výber posledných súborov s ktorými aplikácia pracovala. Po výbere súboru sa mapa budovy vykreslí.

Vykresľovanie mapy je vykonávané pomocou triedy DrawMap. Vzostupne sú prechádzané poschodia mapy a každé je vykresľované do vlastného grafického prvku TabItem. Na vykresľovanie nájdenej trasy slúži trieda DrawTrace, ktorá vykreslí postupne časti trasy do prvkov TabItem

odpovedajúcim poschodiu, na ktorom sa časť nachádza. Obe triedy sa nachádzajú v knižnici tried DrawTypes.



Obr.15 Zobrazenie gridu miestnosti. Naľavo reprezentácia pre HPA* algoritmus s veľkosťou klastra 20, napravo triangulácia priestoru s ktorou pracuje algoritmus A*.

Pri vykonávaní vyhľadávacieho algoritmu je možnosť vizualizovať grid miestnosti, s ktorým aktuálne zvolený vyhľadávací algoritmus pracuje. Toto vykresľovanie vykonávajú metódy DrawHPAGrid pre HPA* algoritmus a DrawTriangulationGrid pre algoritmus A*.

4.3 Implementácia vyhľadávacích algoritmov

Pri implementácii vyhľadávacích algoritmov bolo nutné vybrať algoritmus, ktorý vyhovuje stanoveným podmienkam. Týmito podmienkami boli optimálnosť vyhľadanej cesty, schopnosť vyhľadávania v priestore s prekážkami a rýchlosť algoritmu. Tomuto vyhovovali algoritmy A* (3.2.4) a Hierarchický A* (3.1.2) z toho dôvodu, že pracujú s priestorom ktorý je vopred známy. V nasledujúcich kapitolách priblížim implementáciu týchto algoritmov.

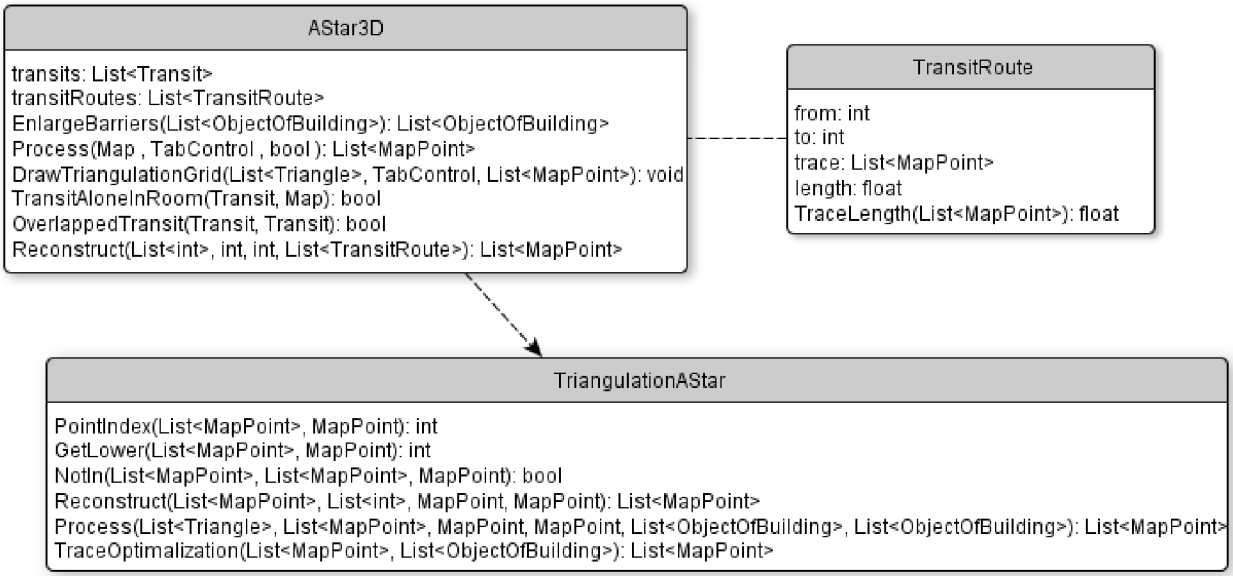
4.3.1 A* algoritmus

Keďže algoritmus A*, ktorý pracuje s priestorom ktorý je reprezentovaný v karteziánskom súradnicovom systéme, je časovo náročný , bolo potrebné priestor predspracovať do jednoduchšej reprezentácie priestoru.

Na toto bola vybraná Delaunayova triangulácia priestoru (2.1.4), ktorej triedy sa nachádzajú v knižnici tried DelaunayTriangulationTypes. Základ tvorí trieda TriangulationGenerator, ktorá triangluláciu vygeneruje ako množinu trojuholníkov typu Triangle. Najprv je generovaný Voronoiov diagram (2.1.3), ktorý pospája priľahlé body. Následne sú z tohto diagramu odstraňované kolízne spoje a vytvárané Delaunayove trojuholníky. Pre ľahšiu manipuláciu s množinou trojuholníkov bola vytvorená trieda Set, ktorá implementuje potrebné funkcie na prácu s množinou. Trieda ConvexPoint reprezentuje konvexný bod, ktorý uchováva informácie o svojich susednostiach a viditeľnosti s ostatnými susednými bodmi.

Samotný algoritmus je rozdelený na dve časti. Prvá časť, ktorá vracia už nájdenú trasu zo štartovacieho do cieľového bodu, je implementovaná v triede Astar3D. Postupne sú spracované všetky miestnosti v budove. Medzi každou dvojicou prechodov v miestnosti je nájdená cesta a uložená do množiny ciest medzi prechodmi typu TransitRoute. Výnimkou sú prechody, ktoré sú v miestnosti osamotené, v miestnosti sa nenachádzajú žiadne iné prechody ani štartovací, či cieľový bod. Pokiaľ sa v danej miestnosti nachádza štartovací alebo cieľový bod, tento je pridaný ako prechod. Do množiny prechodov sú pridané aj párové prechody, teda tie, ktoré spájajú miestnosti medzi sebou. Ich ohodnotenie záleží od vopred udanej konštantnej hodnoty, ktorá vyjadruje zdržanie pri prekonávaní prechodu. Po získaní všetkých ciest medzi prechodmi budovy je použitý A* algoritmus, ktorý medzi nimi vyhľadá cestu. Ako heuristická funkcia je použitá Euklidova vzdialenosť ťažiska prechodu k cieľovému bodu.

Druhú časť tvorí trieda TriangulationAStar, ktorá vyhľadáva cestu medzi prechodmi v miestnosti, ktorá bola spomínaná v predchádzajúcom odseku. Miestnosť je zjednodušená pomocou Delaunayovej triangulácie a následne A* algoritmus vyhľadáva trasu medzi štartovacím a cieľovým prechodom. Po nájdení trasy je cesta optimalizovaná. Každé tri hraničné body trasy sú testované. Pokiaľ existuje priama cesta medzi prvým a posledným bodom, ktorá nie je v kolízii v prekážkou, prostredný bod je z trasy odstránený. Obe časti sú súčasťou knižnice tried AStarTypes.



Obr.16 Class diagram knižnice tried AStarTypes.

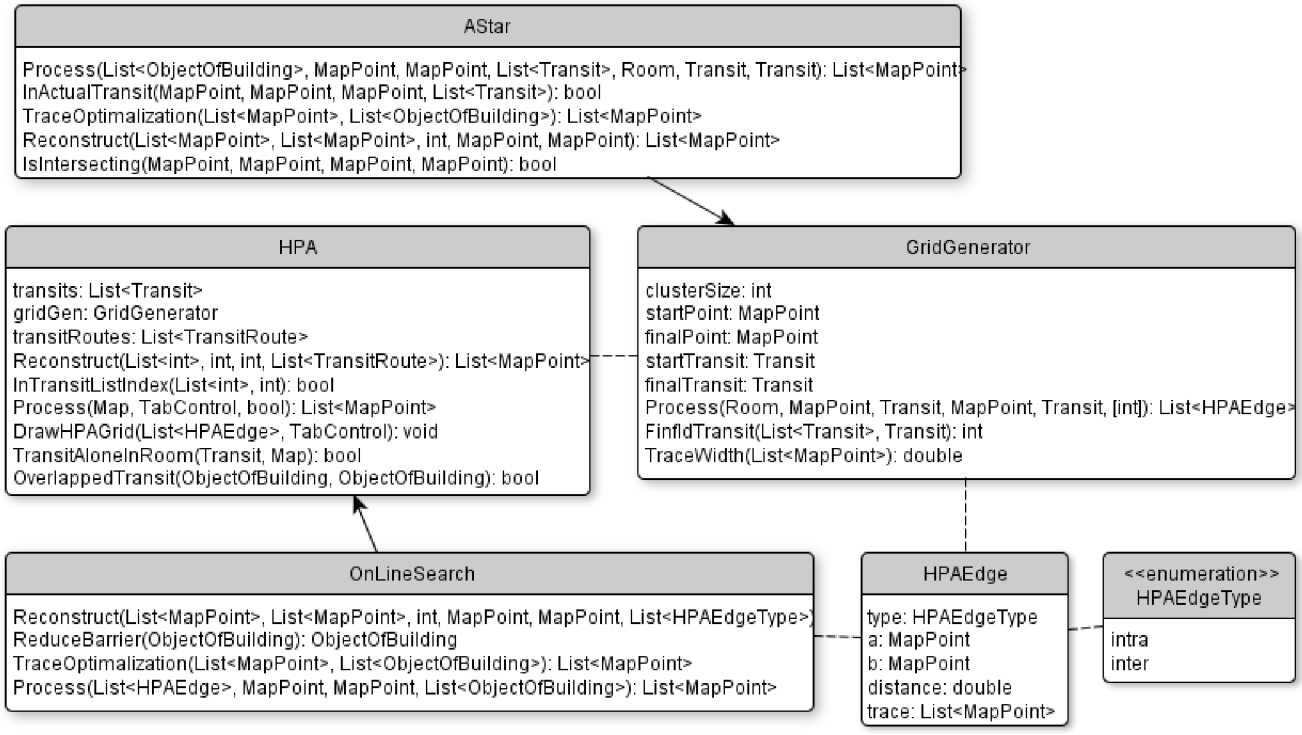
4.3.2 Hierarchický A* algoritmus (HPA*)

Algorimus HPA* funguje na princípe rozdelenia priestoru na klastre, čo sú rovnako veľké polygóny, v našom prípade sú to štvorce. Množinu klastrov generuje trieda GridGenerator. Tu je možné variovať veľkosť generovaných klastrov. Generátor vracia množinu prepojení medzi hraničnými bodmi klastrov. Algoritmus generátoru prechádza miestnosť vodorovne, generuje klustre

a vyhľadáva v nich hraničné body. Pre každé dva susediace hraničné body, ktoré sa náchádzajú v rozdielnych klustroch, je vygenerovaný prechod typu HPAEdge s označením inter. Všetky nájdené hraničné body sú samostatne uchovávané pre každý klaster osobitne. Po vygenerovaní všetkých klastrov sú tieto body spracované. Pre všetky hraničné body náležiacemu jednému klastru sú vyhľadané prepojenia medzi nimi a vytvorený prechod typu HPAEdge s označením intra. Toto vyhľadávanie prepojenia medzi hraničnými bodmi klastra je vykonávané triedou Astar. Táto trieda pracuje s priestorovou reprezentáciou Karteziánskej sústavy súradníc. Aj tu je ako heuristická funkcia použitá Euklidova vzdialenosť dvoch bodov.

Samotné vyhľadávanie trasy medzi štartovacím a cieľovým bodom v budove je podobné ako pri algoritme A* s použitím triangulácie priestoru a je implementované v triede HPA. Pre každú dvojicu prechodov, z ktorých ani jeden nie je osamotený, je vyhľadaná cesta v priestore pomocou triedy OnLineSearch. Táto trieda najprv získa množinu prechodov typu HPAEdge a následne medzi nimi pomocou algoritmu A* vyhľadá trasu zo štartovacieho do cieľového bodu. Po vyhľadaní ciest medzi každými dvoma prechodmi rovnakej miestnosti sú pridané cesty medzi prechodmi, ktoré sú párové, to znamená medzi prechodmi ktoré umožňujú prechody medzi miestnosťami. Namiesto ohodnotenia cesty medzi nimi jej dĺžkou, ohodnotenie týchto prechodov je vopred dané a pre každý typ prechodu rôzne. Cesta je vyhľadávaná naprieč týmito prechodmi, pokiaľ nie je dosiahnutý cieľový bod.

Triedy, ktoré implementujú hierarchický A* algoritmus, sú súčasťou knižnice tried HPATypes.



Obr.17 Class diagram knižnice tried HPATypes.

5 Testovanie

Implementované algoritmy boli testované na mapách rôznych zložitostí. Hlavným testovacím kritériom bola časová náročnosť algoritmov a dĺžka vyhľadanej trasy. Keďže oba algoritmy majú podobný charakter vyhľadávania vo vyššej vrstve algoritmu, to je vyhľadávanie trasy medzi zadaným štartovacím a cieľovým bodom v množine prechodov, ďalšími kritériami boli počet vygenerovaných primitív, jednoduchých geometrických objektov gridu, a počet rozgenerovaných primitív, z tejto množiny, pri vyhľadávaní cesty. Pre každý testovací prípad bola vykonaná séria piatich testov a ich priemerná hodnota bola zapísaná ako výsledná do tabuľky. Veľkosť klastru je uvádzaná v počte buniek. Testovanie prebiehalo na zariadení s parametrami: CPU = Intel(R) Core(TM) Duo T6600 2.20 GHz, RAM = 3GB, Operačný systém = Windows 7 Home Premium 32-bit.

5.1 Výsledky testovania

1. Prvý test bol vykonaný na najjednoduchšej budove, ktorá má jedno poschodie, jednu miestnosť teda aj štartovací a cieľový bod sa nachádzajú v tejto miestnosti, viď príloha 2.

Algoritmus	Veľkosť klastru	Dĺžka cesty	Čas	Počet primitív	Počet rozgenerovaných primitív
A*	-	6570	0,51ms	12	10
HPA*	10x10	6018	0,58s	1281	50
HPA*	20x20	6620	0,42s	369	28
HPA*	30x30	6246	0,40s	192	23

2. Ďalší test bol vykonaný na mape budovy ktorá obsahuje dve miestnosti, ale stále ba jedno poschodie. Štartovací a cieľový bod sú v odlišných miestnostiach, viď príloha 3.

Algoritmus	Veľkosť klastru	Dĺžka cesty	Čas	Počet primitív	Počet rozgenerovaných primitív
A*	-	12319	0,75ms	16	15
HPA*	10x10	13407	0,86s	1759	81
HPA*	20x20	12987	0,61s	509	49
HPA*	30x30	12233	0,58s	270	33

3. Pre nasledujúci test bola zvolená mapa dvojposchodovej budovy. Štartovací a cieľový bod sa nachádzajú na odlišných poschodiach, viď príloha 4.

Algoritmus	Veľkosť klastru	Dĺžka cesty	Čas	Počet primitív	Počet rozgenerovaných primitív
A*	-	12319	0,93ms	22	21
HPA*	10x10	13407	1,19s	2255	115
HPA*	20x20	12987	0,85s	683	73
HPA*	30x30	12233	0,80s	305	48

4. V ďalšom teste bola použitá mapa reprezentujúca zjednodušené bloky O, N a M budovy FIT. Mapa obsahuje dve poschodia a tri jednotlivé budovy prepojené medzi sebou, viď príloha 5.

Algoritmus	Veľkosť klastru	Dĺžka cesty	Čas	Počet primitív	Počet rozgenerovaných primitív
A*	-	26794	17,65ms	124	93
HPA*	10x10	25422	3,31s	7488	652
HPA*	20x20	24824	17,34s	2838	224
HPA*	30x30	25206	65,85s	1905	178

5. Posledné testovanie bolo vykonané na zjednodušenej mape novej budovy FIT, viď príloha 6.

Algoritmus	Veľkosť klastru	Dĺžka cesty	Čas	Počet primitív	Počet rozgenerovaných primitív
A*	-	314377	0,66s	656	412
HPA*	10x10	314339	67,20s	387643	58216
HPA*	20x20	314643	92,56s	15888	2060
HPA*	30x30	312121	284,09s	23403	2516

5.2 Zhodnotenie výsledkov

V nasledujúcom texte zhodnotím výsledky testovania, ktoré boli predstavené v predchádzajúcej kapitole. Všetky testované algoritmy, tak ako A* algoritmus, tak aj HPA* algoritmus s rôznymi veľkosťami klastru, našli cestu zo štartovacieho do cieľového bodu. Pri troch z piatich testovaných máp našiel najkratšiu cestu algoritmus HPA* s veľkosťou klastru 30x30 buniek a teda na základe výsledkov je optimálny. Ako druhý, čo sa dĺžky vyhľadanej cesty týka, bol A* algoritmus s Delaunayovou trianguláciou priestoru.

Druhým hlavným kritériom pri testovaní bola časová náročnosť. Vo všetkých prípadoch testovaných máp najrýchlejšie pracoval algoritmus A*, kde sa jeho časová náročnosť pohybovala v intervale menšom ako je jedna sekunda. Druhým najrýchlejším algoritmom bol HPA*, s veľkosťou klastru 30x30 buniek, ktorý bol pri prvých troch testovacích mapách najrýchlejší. Problém pre tento algoritmus nastáva v mapách, ktoré obsahujú väčší počet miestností, teda nad 10 a v mapách, ktorých miestnosti majú menšie rozmery ako 30x30 buniek. V týchto prípadoch časová zložitosť rýchlo rastie.

Poslednými kritériami, na ktoré som sa pri testovaní zameral, bol počet primitív, ktoré pri zjednodušovaní priestoru mapy algoritmus generuje a počet primitív z tejto generovanej množiny, ktoré následne pri hľadaní cesty znovu spracuje. Algoritmus A* generoval pri všetkých testovaných mapách najmenej primitív a takisto ich aj najmenej spracoval pri hľadaní cesty. Výsledky algoritmu HPA* pri týchto dvoch kritériách, poukazujú na fakt, že čím menšie sú rozmery klastru v generovanom gride tým väčšie sú pamäťové nároky tohto algoritmu. Výnimku v tomto tvrdení

tvoria mapy, ktorých rozmery miestností sú menšie ako veľkosť jedného klastru. Toto môžeme vidieť na výsledkoch piateho testu.

Keďže v troch zo štyroch testovacích kritérií bol algoritmus A* najlepšie hodnotený, tento algoritmus považujem ako najlepší pre riešenie problematiky tejto bakalárskej práce.

6 Záver

V tejto práci som sa zameral na vyhľadávacie algoritmy, ktoré sú schopné nájsť cestu v 3D priestore a na reprezentácie priestoru, s ktorými tieto algoritmy pracujú. Práca popisuje niektoré reprezentácie reálneho priestoru a reprezentácie stavového priestoru. Následne vysvetľuje vybrané vyhľadávacie algoritmy.

Bola navrhnutá aplikácia, ktorá implementuje vybrané metódy pre vyhľadávanie v priestore budovy. Tými sú A* algoritmus pracujúci s Delaunayovou triangulációu priestoru a Hierarchický A* algorimus. V aplikácii je možné zvoliť algoritmus, ktorý vyhľadá cestu zo zvoleného štartovacieho bodu do cieľového bodu. Aplikácia taktiež obsahuje objektovú reprezentáciu budovy. Pre možnosť vytvárať a ukladať mapy budovy v tejto objektovej reprezentácii, bola vytvorená jej XML verzia a popis v DTD formáte tohto XML pre možnosť jeho vytvorenia.

Implementované algoritmy boli testované na rôznych typoch budov, ktorých zložitosť sa líšila. Pri zhodnotení výsledkov testovania bolo zistené, že na riešenie problematiky vyhľadávania cesty v budove je najvhodnejší algoritmus A* s Delaunayovou triangulációu priestoru.

Literatúra

[1] VOJTEKOVÁ, Mária. přednáška z předmětu Geometria. *Súradnicové sústavy*. Žilina, 2012. Dostupné z: http://fpedas.uniza.sk/~vojtek/21_SurSustavy.pdf

[2] CORMEN, Thomas H. *Introduction to algorithms*. 2nd ed. Cambridge: MIT Press, c2001, xxi, 1180 s. ISBN 02-620-3293-7.

[3] GEVA, S. K-tree: a height balanced tree structured vector quantizer. *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)* . IEEE, 2000, s. 271-280.

[4] LAVALLE, Steven Michael. *Planning algorithms*. 1st ed. New York: Cambridge University Press, 2006, xvi, 826 s. ISBN 05-218-6205-1.

[5] Simple Stupid Funnel Algorithm. In: MONONEN, Mikko. *Digesting Duck*: *Blog about game AI and prototyping* [online]. 2010, 8. Marec 2010 [cit. 2013-01-15]. Dostupné z: http://digestingduck.blogspot.cz/2010/03/simple-stupid-funnel-algorithm.html

[6] DEMYEN, Douglas Demyen a Michael BURO. Efficient Triangulation-Based Pathfinding. University of Alberta. 2005, 6 s.

[7] LIKHACHEV M., FERGUSON D., GORDON G., STENTZ A., THRUN S.,"Anytime Dynamic A*: An Anytime, Replanning Algorithm," *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June, 2005.

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

[9] KELLNHOFER, Petr. *K-d tree a bd-tree pro dělení prostoru*. Plzeň, 2012. Semestrální práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky, Centrum počítačové grafiky a vizualizace dat.

[10] BOTEA, A, M MÜLLER a J SCHAEFFER. Near Optimal Hierarchical Path-Finding. *Journal of game development*. 2004, Volume 1, Issue 1, s. 7-28. ISSN 1543-9399.

[11] KOENIG, Sven, Maxim LIKHACHEV a David FURCY. Lifelong Planning A* . *Artificial Intelligence*. 2004, roč. 155, 1-2, s. 93-146. ISSN 00043702.

[12] PÉTER, Ján. *Iteračne generované textúry*. Bratislava, 2006. Diplomová práce. Univerzita Komenského v Bratislave. Vedoucí práce Doc. RNDr. Andrej Ferko PhD.

[13] JOLLIFFE, Ian. *Principal Component Analysis*, Springer Series in Statistics, Springer 2002, 2nd Edition, ISBN 0387954422, 9780387954424

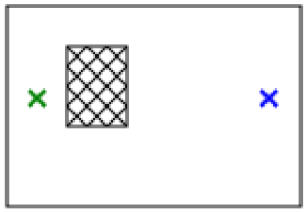
[14] MICROSOFT CORPORATION. 2013. Introduction to WPF. Windows Presentation Foundation. Dostupné z: http://msdn.microsoft.com/en-us/library/aa970268.aspx

Zoznam príloh

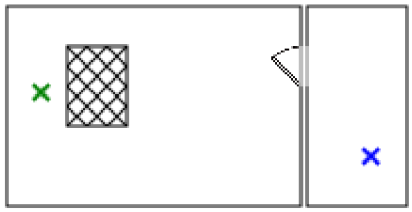
Príloha 1 - DTD formát XML súboru popisujúceho objektovú reprezentáciu budovy

```
<?xml encoding="UTF-8"?>
<!ELEMENT Map (Floors,Start,Final)>
<!ATTLIST Map xmlns CDATA #FIXED ''>
<!ELEMENT Floors (Floor)+>
<!ATTLIST Floors xmlns CDATA #FIXED ''>
<!ELEMENT Start (X,Y,Z)>
<!ATTLIST Start xmlns CDATA #FIXED ''>
<!ELEMENT Final (X,Y,Z)>
<!ATTLIST Final xmlns CDATA #FIXED ''>
<!ELEMENT Floor (Rooms,FloorNumber)>
<!ATTLIST Floor xmlns CDATA #FIXED ''>
<!ELEMENT Rooms (Room)+>
<!ATTLIST Rooms xmlns CDATA #FIXED ''>
<!-- FloorNumber have to be number from interval 0..(Floor count - 1) -->
<!ELEMENT FloorNumber (#PCDATA)>
<!ATTLIST FloorNumber xmlns CDATA #FIXED ''>
<!ELEMENT Room (Corners,Barriers,Transits,Name)>
<!ATTLIST Room xmlns CDATA #FIXED ''>
<!ELEMENT Barriers (ObjectOfBuilding)*>
<!ATTLIST Barriers xmlns CDATA #FIXED ''>
<!ELEMENT Transits (Transit)*>
<!ATTLIST Transits xmlns CDATA #FIXED ''>
<!ELEMENT Name (#PCDATA)>
<!ATTLIST Name xmlns CDATA #FIXED ''>
<!ELEMENT ObjectOfBuilding (Corners)>
<!ATTLIST ObjectOfBuilding xmlns CDATA #FIXED ''>
<!ELEMENT Transit (Corners,Delay,Id,Type)>
<!ATTLIST Transit xmlns CDATA #FIXED ''>
<!-- Delay have to be number that indicates price of passing transit -->
<!ELEMENT Delay (#PCDATA)>
<!ATTLIST Delay xmlns CDATA #FIXED ''>
<!ELEMENT Id (#PCDATA)>
<!ATTLIST Id xmlns CDATA #FIXED ''>
<!ELEMENT Type transitType (nonSet|door|elevator|stairs) "nonSet">
<!ATTLIST Type xmlns CDATA #FIXED ''>
<!ELEMENT Corners (MapPoint)+>
<!ATTLIST Corners xmlns CDATA #FIXED ''>
<!ELEMENT MapPoint (X,Y,Z)>
<!ATTLIST MapPoint xmlns CDATA #FIXED ''>
<!-- X , Y, Z have to be number that indicates x-position at space -->
<!ELEMENT X (#PCDATA)>
<!ATTLIST X xmlns CDATA #FIXED ''>
<!ELEMENT Y (#PCDATA)>
<!ATTLIST Y xmlns CDATA #FIXED ''>
<!ELEMENT Z (#PCDATA)>
<!ATTLIST Z xmlns CDATA #FIXED ''>
```

Príloha 2 - Mapa budovy použitá v 1. teste

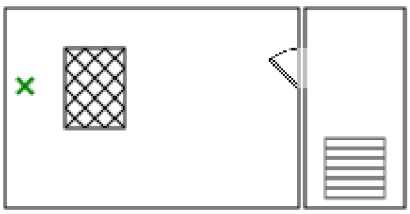


Príloha 3 - Mapa budovy použitá v 2. teste



Príloha 4 - Mapa budovy použitá v 3. teste

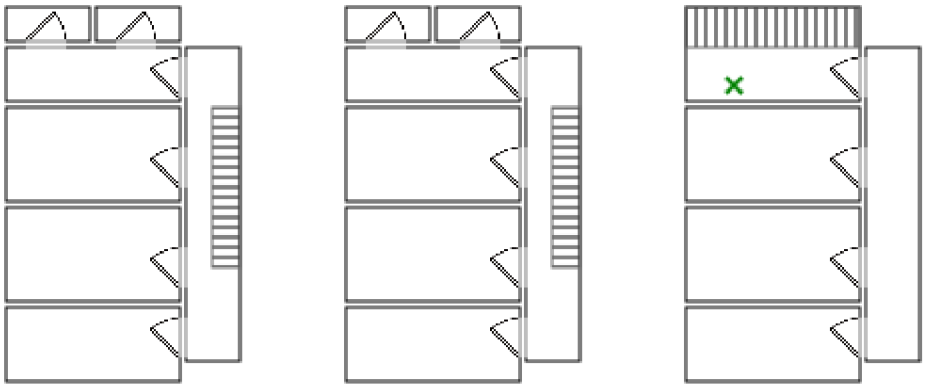
1. poschodie



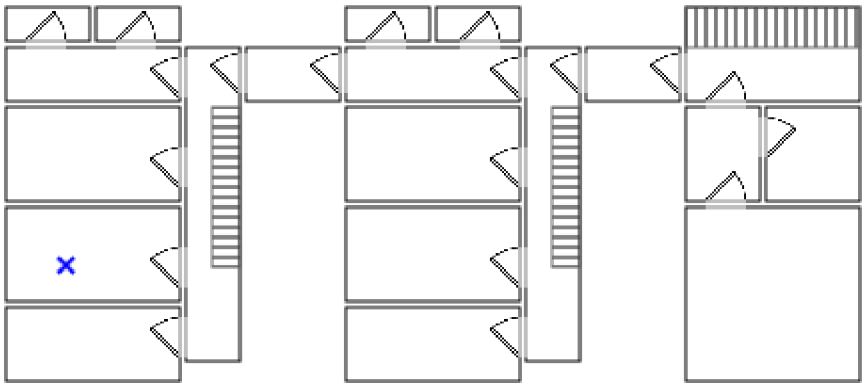
2. poschodie

Príloha 5 - Mapa budovy použitá vo 4. teste

1. poschodie

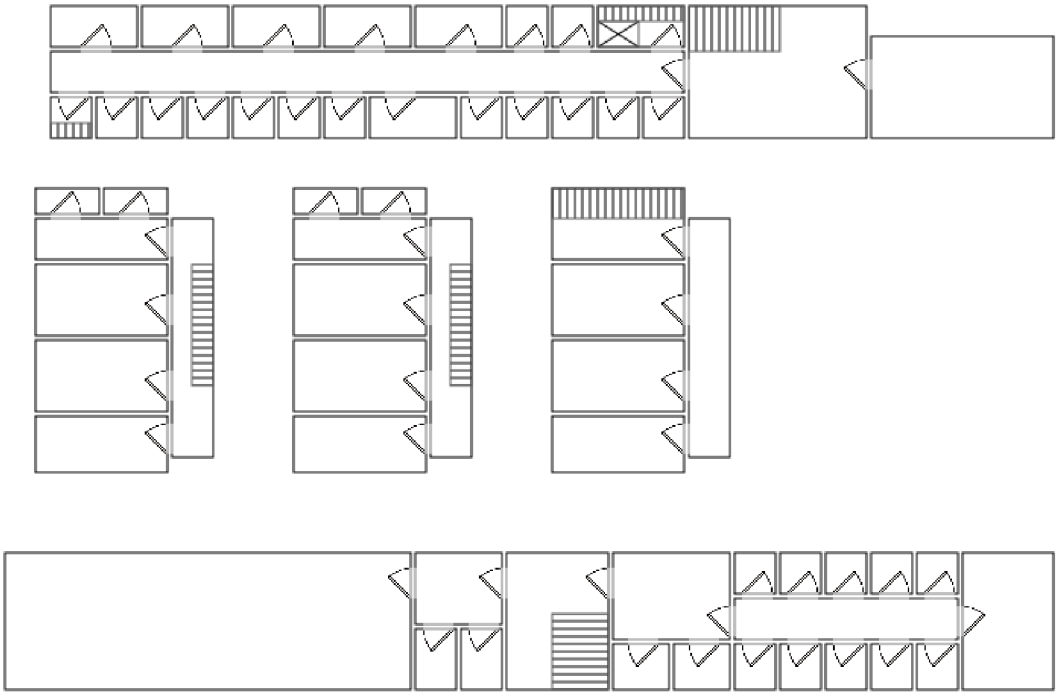


2. poschodie

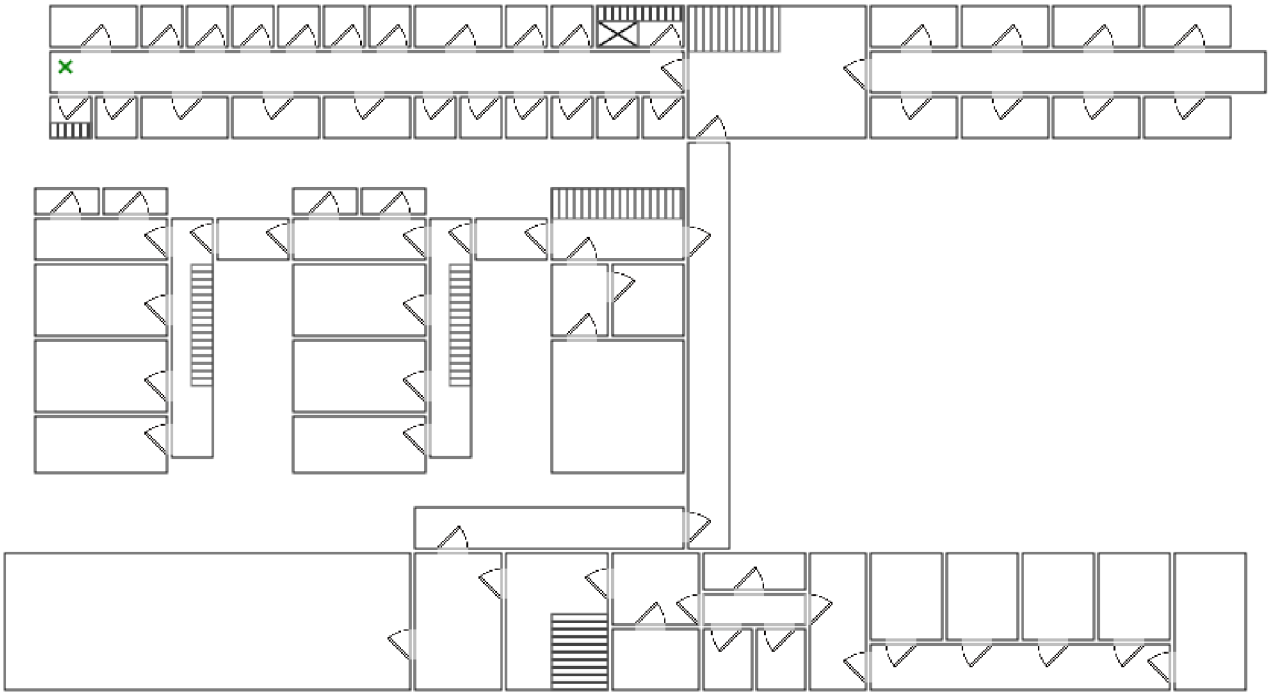


Príloha 6 - Mapa budovy použitá v 5. teste

1. poschodie



2. poschodie



3. poschodie