VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

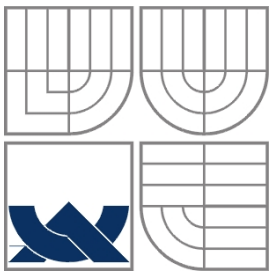
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYHLEDÁVÁNÍ OPTIMÁLNÍ CESTY TERÉNEM

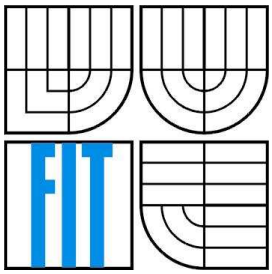
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JAN VÁŇA
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYHLEDÁVÁNÍ OPTIMÁLNÍ CESTY TERÉNEM

SEARCH FOR THE OPTIMAL PATH IN THE TERRAIN

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JAN VÁŇA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN DRAHANSKÝ Ph.D.
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Váňa Jan**
Obor: Informační technologie
Téma: **Vyhledávání optimální cesty terénem**
Kategorie: Umělá inteligence

Pokyny:

1. Nastudujte literaturu se zaměřením na vyhledávání optimální cesty (umělá inteligence).
2. Navrhněte postup realizace vyhledávání optimální cesty terénem (např. vyhýbání se překážkám apod.), který bude vhodný pro databázi (3D prostředí) firmy E-COM.
3. Navržený postup prakticky implementujte.
4. Proveďte experimenty a dosažené výsledky zhodnoťte.

Literatura:

- Dle specifikace školitele a průmyslového zadavatele.

Při obhajobě semestrální části projektu je požadováno:

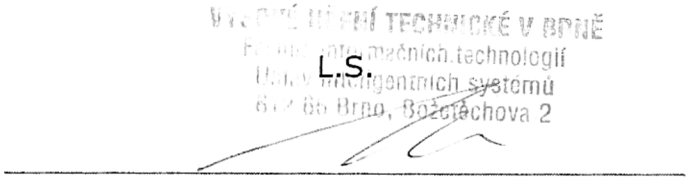
- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Drahanský Martin, Ing., Dipl.-Ing., Ph.D.**, UITS FIT VUT
Konzultant: Janeček Petr, Ing., E-COM
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007



doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Jan Váňa**
Id studenta: 84100
Bytem: Zahradní 294, 281 21 Červené Pečky
Narozen: 06. 07. 1985, Kolín
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Vyhledávání optimální cesty terénem
Vedoucí/školitel VŠKP: Drahanský Martin, Ing., Dipl.-Ing., Ph.D.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Simulační systémy využívající prvků umělé inteligence jsou často stavěny před problém, jak vyhledat cestu v terénu pro libovolný objekt řízený počítačem. Tento projekt se zabývá tvorbou modulu začlenitelného do jakékoli aplikace, jenž dokáže tento problém vyřešit. Pro určitý objekt je schopen najít optimální cestu do cílového místa pokud taková cesta existuje.

Klíčová slova

vyhledávání cest, prohledávání grafů, A*, umělá inteligence

Abstract

Simulation systems based on artificial intelligence often have to solve the problem of finding a way between two places for an object controlled by a computer. This article deals with a development of such a module for already existing simulation system, which is able to find the best way for any object if this is existent.

Keywords

path-finding, graph traversal, A-star, artificial intelligence

Citace

Jan Váňa: Vyhledávání optimální cesty terénem, bakalářská práce, Brno, FIT VUT v Brně, 2007

Vyhledávání optimální cesty terénem

Prohlášení

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

……
Jan Váňa
3. května 2007

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu práce Ing. Martinovi Drahanskému Ph.D. za zájem, připomínky a hlavně za čas, který věnoval mé práci.
Dále patří poděkování Ing. Petru Janečkovi a společnosti E-COM s.r.o. za poskytnutí zajímavého zadání a pomoc při řešení práce.

Obsah

Úvod

Automatické vyhledávání cest je problém, kterým se v dnešní době zabývá stále větší počet odvětví. Ve vojenství se jednotkám může hodit znalost nejkratší, nebo nejbezpečnější cesty do cíle. V robotice se může robot nezávisle rozhodnout kudy se nejlépe dostane na určené místo i bez dozoru člověka. Hledání cest je užitečné i v dopravě ať už pro plánování trasy, nebo pro automaticky řízená vozidla. Další nezanedbatelnou oblastí, která se touto problematikou zabývá, jsou simulace a počítačové hry, tedy odvětví, v nichž se objekt nepohybuje v reálném světě, ale v čistě virtuálním prostředí. Využití hledání cest je opravdu široké.

Cílem mého projektu je vytvořit samostatný modul, který bude schopen automaticky vyhledat (optimální) cestu mezi libovolnými dvěma body v terénu. Tento modul by měl být snadno začlenitelný do systému, jenž tuto funkčnost požaduje.

V této práci je popsán průběh návrhu, vývoje a testování celého projektu. V první kapitole naleznete informace o cílech a zadání projektu. Další část se na problematiku dívá z teoretického hlediska, objasňuje pojmy a některé základní postupy. Třetí kapitola je věnována návrhu řešení celého projektu, zejména volbě algoritmů a tvorbě mapové a grafové reprezentace. Kapitola číslo 4 zmiňuje způsob samotného řešení, jak bylo v projektu nakonec implementováno. Předposlední část práce popisuje různá vylepšení, která by měla znatelně zvýšit kvalitu celého modulu. V závěrečné kapitole jsou shrnuty výsledky provedených testů.

1 Cíle projektu

1.1 Účel vytvoření

Společnost, pro níž je projekt vytvářen, se zabývá vývojem simulátorů z vojenského i civilního prostředí. V současné době jsou všechny simulace prováděny podle předem daného postupu. Existuje předpis, který říká, co se ve které chvíli bude dít, a každé spuštění simulace podle daného předpisu tedy proběhne totožně. Novým rozšířením stávajícího systému je zakomponování automaticky řízené simulace. Umělá inteligence počítače by měla sama rozhodovat o tom, co se bude dále dít v závislosti na chování v prostředí umístěných objektů.

Z tohoto důvodu není možné předem naplánovat trasu, po které se budou automaticky řízené objekty během simulace pohybovat, a je tedy nutné řešit úlohu hledání cesty mezi libovolnými místy na mapě.

1.2 Analýza požadavků

Cílem celého projektu je tedy vytvoření modulu, který půjde snadno zakomponovat do konkrétního simulátoru. Modul bude schopen vyhledat cestu a do systému zpátky poslat informace o tom, zda hledání bylo úspěšné, a v případě že bylo, vrátí poznatky o vypočítané cestě.

1.2.1 Vstupy

Modul musí být schopen pracovat s mapou, nad kterou simulace běží. Proto je základním vstupem modulu předání informací o mapě. Je vhodné, aby se modul sám ptal na konkrétní údaje, než aby simulační systém rozhodoval o předání dat do modulu.

Pro vlastní vyhledávání je třeba znát počáteční a koncový bod cesty. Předání těchto údajů je triviální záležitostí.

Poslední podmínkou pro správné vyhledání je předání správných parametrů, jakými jsou například rozměry objektu, jeho schopnost pohybovat se v prostředí a také například to, jaké jsou požadavky na nalezení cesty (co nejrychleji, co nejpřesněji, orientačně, …).

1.2.2 Výstupy

Na výstupu modulu simulátor očekává základní poznatek o úspěchu či neúspěchu vyhledávání. V případě, že cesta byla nalezena, předá modul vektor souřadnic, přes který cesta prochází. To, do jaké míry je výsledný vektor „jemný“, záleží na parametrech vyhledávání. Někdy dokonce modul může vracet jen část cesty, v tom případě připojí informaci o tom, že cesta není kompletní.

1.2.3 Volba implementačního jazyka

Celý stávající systém je napsán v programovacím jazyce C++, proto je logické, že projekt je tvořen ve stejném jazyce. Usnadní to integraci a zvýší se rychlost komunikace mezi modulem a simulátorem.

1.3 Obecné použití modulu

Celý modul je sice vytvářen primárně pro nasazení v jednom konkrétním simulačním prostředí, nicméně má smysl jej vyvíjet co nejobecněji, aby se dal pouze s minimálními úpravami použít i v jiných aplikacích. Platí ale zásada, že efektivnost a co nejlepší funkčnost modulu při nasazení do vyvíjeného prostředí má přednost před obecností. Tuto zásadu je třeba zohlednit v případech, kdy neexistuje řešení vyhovující oběma podmínkám.

2 Teoretický rozbor

2.1 Formalizace

Pro vyhledání cesty s použitím počítače musíme problém vyhledávání cesty v terénu nejdříve formalizovat. To znamená převést ho do podoby, ve které je ho schopen počítač vyřešit. Nejpodobnější úloha, která je dobře popsaná a s pomocí počítače snadno řešitelná, je úloha hledání nejkratší cesty v ohodnoceném grafu. Musíme tedy vymyslet způsob, jak základní úlohu na tento problém transformovat. Nejprve si definujeme několik základních pojmů se kterými se v textu pracuje.

2.1.1 Graf

Graf G je uspořádaná dvojice (U, H), kde U je neprázdná množina uzlů grafu a H množina některých dvojic prvků (hran) z množiny U. Skutečnost, že hrana h spojuje uzly u_1 a u_2, označme jako $h = \{u_1,u_2\}$ [WIKI].

2.1.2 Ohodnocený graf

Zachováme-li značení z předchozí definice, pak ohodnocený graf je graf, pro který platí, že každému prvku h z množiny H je přiřazeno ohodnocení $c(h)$ [WIKI].

2.1.3 Cesta v grafu

Termínem cesta v grafu $G = (U, H)$ označuje posloupnost $P = (u_0, h_1, u_1, \ldots, h_n, u_n)$, kde $h_i = \{u_{i-1}, u_i\}$ a navíc $u_i \neq u_j$ pro libovolné $i \neq j$.
Je to tedy posloupnost uzlů, pro kterou platí, že v grafu existuje hrana z daného uzlu do jeho následníka. Žádné dva uzly (a tedy ani hrany) se přitom neopakují [WIKI].

2.1.4 Cesta z uzlu A do B

Cesta v grafu $G = (U, H)$ z uzlu A do B je taková cesta $P_{(A,B)} = (u_0, h_1, u_1, \ldots, h_n, u_n)$, která splňuje následující: $u_0 = A$ a $u_n = B$.

2.1.5 Ohodnocení cesty v ohodnoceném grafu

Ohodnocení C cesty $P = (u_0, h_1, u_1, \ldots, h_n, u_n)$ v ohodnoceném grafu se spočítá jako součet všech dílčích ohodnocení hran $c(h_i)$ v cestě P.

2.1.6 Nejkratší cesta v grafu

Je to taková cesta $P_{i(A,B)}$, která splňuje, že z množiny M všech cest z počátečního uzlu A do cílového uzlu B, je $C(P_i)$ minimální.

2.2 Algoritmizace problému

Pro řešení problému vyhledávání nejkratší cesty v grafu existuje řada vyzkoušených algoritmů [IZU]. Jediným problémem je tedy samotný převod mapy na ohodnocený graf.

2.3 Převedení mapy na ohodnocený graf

Jak již bylo výše zmíněno, celou mapu musíme převést na ohodnocený graf, aby bylo úlohu možné vyřešit pomocí známých a vyzkoušených algoritmů, určených pro prohledávání grafů. To znamená, že se z dostupných informací o mapě, musíme nějakým způsobem získat údaje o uzlech grafu a o existenci a ohodnocení hran mezi nimi. Existují dva základní způsoby jak toho dosáhnout [AMIT]. Oba způsoby se liší v tom, s jakou reprezentací mapy pracují. První způsob předpokládá, že máme k dispozici vektorový popis mapy (pokud neexistuje, musíme takový popis získat), z kterého se graf vytvoří. Druhý způsob naopak vytváří graf z rastrového popisu mapy (mapa je uložena jako matice políček).

2.3.1 Vektorový popis mapy

Celá mapa se skládá z bodů, úseček a polygonů. Pro vytvoření grafu z takovéto mapy se nabízí mapovat všechny vrcholy úseček a polygonů přímo na uzly grafu. Hrany pak spojují pouze ty uzly, které reprezentují místa, jenž jsou vzájemně dostupné (úsečka, která je spojuje, neprotíná žádný existující objekt). Počet hran spojujících daný uzel s některým jiným uzlem je tedy shora omezen maximálním počtem uzlů v grafu mínus jedna.

Pokud se zachová poměr mapování 1:1, pak by měl být vektorový popis mapy ekvivalentní vytvořenému grafu. Problematika vytvoření grafu pro účely vyhledávání cesty z vektorového popisu mapy [NA] je velmi obsáhlá a bere se v ní v úvahu například i velikost objektu, pro který cestu budeme vyhledávat, úhly mezi vektory, atd.

2.3.2 Rastrový popis mapy

Rastrový popis (dvou-dimenzionální) mapy lze chápat jako množinu $M = X \times Y$ (dvojrozměrná síť políček). Každý prvek této množiny je charakterizován souřadnicemi $[x,y]$, pro které platí vztahy (2.1) a (2.2).

$$0 \leq x < X \tag{2.1}$$

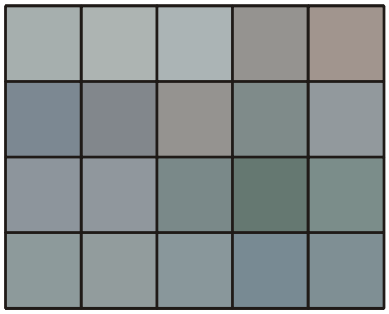
$$0 \leq y < Y \tag{2.2}$$

Políčka *A* a *B* spolu sousedí právě tehdy když platí vztah (2.3). To znamená, že každé políčko může mít maximálně 8 sousedů.

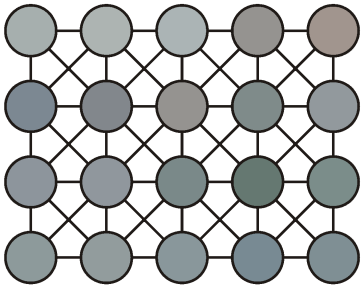
$$|A_x\text{-}B_x| \leq 1 \;\&\; |A_x\text{-}B_x| \leq 1 \;\&\; A_x \neq B_x \;\&\; A_y \neq B_y \tag{2.3}$$

Ohodnocený graf se z tohoto popisu udělá snadno. Každý uzel grafu odpovídá právě jednomu políčku z množiny *M* a hrany spojují uzly reprezentující vzájemně sousedící políčka.

Obrázek 2.1: Původní mapa



Obrázek 2.2: Rastrová reprezentace



Obrázek 2.3: Grafová reprezentace

2.3.3 Ohodnocení hrany grafu

Přiřazení co nejpřesnějšího ohodnocení hrany *h* spojující uzly je nezbytnou podmínkou pro správnou funkčnost celého vyhledávání. Označme uzly, které hrana spojuje, jako u_1 a u_2 a místa na mapě, které jsou těmito uzly reprezentovány, jako m_1 a m_2. Ohodnocení *c(h)* by mělo brát v úvahu skutečnou vzdálenost na mapě mezi místy m_1 a m_2 (označme ji jako $|m_1,m_2|$) a také obtížnost přesunu v terénu mezi těmito dvěma místy. Ohodnocení musí splňovat podmínku (2.4), z důvodů uvedených v kapitole 2.4.

$$c(h) \geq |m_1,m_2| \tag{2.4}$$

Ohodnocení *c(h)* vypočítáme podle vztahu (2.5), kde konstanta *k* vyjadřuje obtížnost průchodu mezi sousedními políčky. Její výpočet bude popsán později.

$$c(h) = k \cdot |m_1,m_2|,\ k \geq 1 \tag{2.5}$$

2.4 Prohledávání grafu

Možností jak řešit úlohu prohledávání grafu, potažmo stavového prostoru, existuje velmi mnoho. Pro řešení úlohy, kdy se má najít optimální cesta v grafu, jsou vhodné pouze ty, které vždy najdou nejlepší řešení, pokud takové existuje. Mezi ně patří následující 3 algoritmy:

2.4.1 Prohledávání do šířky (Breath-first-search)

Základním algoritmem pro prohledávání stavového prostoru je prohledávání do šířky. Principielně funguje tak, že se za aktuální uzel zvolí výchozí místo a projdou se všechna sousedící místa (následníci). Pokud mezi nimi není koncový uzel, pak se postupně volí za aktuální uzly všechna tato sousedící místa, a proces se opakuje. Značnou nevýhodou je že neexistuje žádná priorita při vybírání dalších uzlů, které budou prohlášeny za aktuální, takže se z výchozího stavu graf prohledává do všech stran rovnoměrně.

Algoritmus [IZU]:

1. Existuje seznam OPEN. Vlož do něj počáteční uzel.
2. Je-li OPEN prázdný, úloha nemá řešení.
3. Vyber z OPEN první uzel.
4. Je-li uzel uzlem cílovým, úloha má řešení (vrať posloupnost uzlů).
5. Vybraný uzel expanduj (nalezni všechny sousedy), umísti následníky do OPEN a pokračuj na bodu 2.

Pro daný algoritmus existuje celá řada vylepšení, nicméně žádné neodstraňuje jeho hlavní nevýhody (pomalost, velký počet prozkoumaných uzlů).

2.4.2 Prohledávání podle heuristiky (Best-first-search)

Základní slabinou předchozího algoritmu je fakt, že nijak nevyužívá dostupné informace o tom, kde se nachází počáteční, aktuální a cílový uzel. Pokud lze takové údaje získat, je možno sestavit následující ohodnocující funkce:

- *g(n)* – ohodnocení cesty mezi aktuálním a počátečním uzlem *n* (toto ohodnocení je přesné – lze ho počítat postupně při rozgenerování uzlů – viz kapitola 2.3.3)
- *h(n)* – ohodnocení cesty z aktuálního uzlu *n* do cíle (lze pouze odhadnout na základě nějaké znalosti o daném grafu - např. jedná-li se o mapu, může být tímto ohodnocením přímá spojnice obou bodů)
- *f(n)* = g*(n)* + *h(n)* - heuristická funkce - dává představu o tom, jak nákladná bude asi cesta z počátku do cíle, povede-li přes uzel *n*

Algoritmus je založený na tom, že k expanzi vybíráme nejdříve uzly s nejnižším ohodnocením *f(n)*, jelikož lze očekávat, že hledaná optimální cesta povede právě přes tyto uzly.

Navíc pokud před hodnoty funkcí *g(n)* + *f(n)* předřadíme konstanty, pak můžeme snadno dát větší váhu jedné z funkcí. Algoritmus se tak stává flexibilnějším, i když ztrácí některé své základní charakteristiky.

2.4.3 A * (A-star)

Od algoritmu Best-first-search se liší pouze v tom, že *h(n)* (hodnotící funkce) musí být spodním odhadem skutečné ceny. Lze dokázat [IZU], že pokud je tato podmínka splněna, pak je nalezená cesta optimální. V té nejjednodušší podobě funguje algoritmus takto:

1. Sestroj seznam OPEN a umísti do něj počáteční uzel.
2. Je-li OPEN prázdný, úloha nemá řešení.
3. Vyber z OPEN uzel s nejnižším ohodnocením.
4. Je-li uzel uzlem cílovým, úloha má řešení (vrať posloupnost uzlů).
5. Vybraný uzel expanduj (nalezni všechny sousedy), umísti následníky do OPEN, pokud je zde některý uzel vícekrát, ponechej v OPEN jen ten s nižším ohodnocením a pokračuj na bodu 2.

Tento postup lze vylepšit tak, že se vedle seznamu OPEN sestrojí ještě seznam CLOSE, ten obsahuje již expandované uzly. To umožní snadnou rekonstrukci výsledné cesty a zabrání opakované expanzi uzlů. Finální verze algoritmu pak vypadá takto:

1. Sestroj seznamy CLOSE (obsahuje expandované uzly) a OPEN (obsahuje uzly určené k expanzi) a umísti do OPEN počáteční uzel.
2. Je-li OPEN prázdný, úloha nemá řešení.
3. Vyber z OPEN uzel s nejnižším ohodnocením a vlož ho do CLOSE.
4. Je-li vybraný uzel uzlem cílovým, úloha má řešení (vrať posloupnost uzlů).
5. Vybraný uzel expanduj (nalezni všechny sousedy). Pokud je již některý ze sousedů v OPEN nebo CLOSE s vyšším ohodnocením, odstraň ho z příslušné fronty. Umísti všechny následníky, kteří nejsou v CLOSE ani v OPEN do OPEN a pokračuj od bodu 2.

2.5 Heuristika

Nasazení algoritmu prohledávání podle heuristiky, nebo A* tedy požaduje vyřešit problém výpočtu ohodnocení pro každý uzel grafu. To se pro uzel *n* skládá z ohodnocení (ceny) již absolvované cesty *g(n)* a z odhadu ceny zbývající části cesty *h(n)*.

2.5.1 Ohodnocení absolvované cesty

Toto ohodnocení lze snadno určit pro každý uzel, který je následníkem některého expandovaného uzlu. Platí, že *g(n)* počátečního uzlu je nulové. Označíme-li expandovaný uzel jako *n*, jeho následníky jako n_i a hranu mezi *n* a n_i jako *h*, pak pro $g(n_i)$ platí vztah (2.6).

$$g(n_i) = g(n) + c(h) \tag{2.6}$$

2.5.2 Odhad ohodnocení zbývající cesty

Pro metodu A* je nutné určit spodní odhad výsledného ohodnocení, tedy takové číslo, které je za všech okolností nižší než skutečná cena. Pro naši úlohu je tímto číslem jistě přímá vzdálenost na mapě mezi místy, které reprezentuje aktuální a cílový uzel. Toto lze tvrdit pouze za předpokladu, že ohodnocení hrany mezi sousedními uzly je vždy větší nebo rovno skutečné vzdálenosti mezi políčky, které jsou danými uzly zastoupeny. Pokud je ohodnocení grafu určeno vztahem (2.5), pak je tato podmínka splněna.

3 Navržené řešení

Celé řešení lze rozdělit do dvou hlavních částí. První je tvorba mapové reprezentace a její převedení na ohodnocený graf. Druhou částí je vyhledání samotné cesty grafem.

3.1 Mapová reprezentace

Při volbě mapové reprezentace hraje roli typ map, pro které se má modul používat. Každý způsob reprezentace se hodí pro jiné typy terénu. Vektorový popis mapy lze s výhodou použít u pravidelných geometrických map, kde jsou všechny polygony, z nichž je mapa tvořena, srovnatelně velké s velikostí objektu, nebo ještě lépe větší. Typickým případem jsou interiéry budov, jenž jsou tvořeny velkými obdélníky z homogenního materiálu (chodby, místnosti, …). Naopak nehodí se příliš na výškově i terénně členité mapy, kde je povrch nepravidelně zakřivený a kde se střídají různé typy materiálů (exteriéry, přírodní scenérie, …). Zde se hodí spíše rastrový popis.

Jelikož simulátor, pro který je modul vyvíjen, pracuje většinou s venkovními mapami, zvolil jsem rastrový popis map.

3.2 Rastrová mapa

3.2.1 Tvorba mapy

Vytvoření rastrové mapy, která co nejlépe odpovídá skutečné mapě, v níž se vyhledává, je nutnou podmínkou pro dosažení kvalitních výsledků. Pokud popis mapy neodpovídá skutečnosti, nemá význam se pokoušet o vyhledávání cesty, jelikož informace, které bychom výpočtem získali, by byly bezcenné.

Rastrová mapa se skládá z políček o konstantní velikosti. Při tvorbě mapy se celý povrch, který každé políčko pokrývá, musí převést na několik diskrétních údajů, reprezentujících základní vlastnosti dané plochy (výška, materiál, dostupnost, …) a to tak, aby došlo k co nejmenší chybě.

Je zřejmé, že způsobů jak vytvořit síť políček a přiřadit jí hodnoty je velmi mnoho. Závisí na informacích, které máme o reálné mapě, na typu aplikace, pro který chceme vyhledávat, na požadované přesnosti apod. Například údaj o výšce políčka lze získat buď přímo jako hodnotu ve středu políčka, nebo aritmetickým průměrem výšek v rozích políčka.

Volba velikosti políčka a jeho správná inicializace hraje rozhodující vliv na přesnost a rychlost celého vyhledávacího procesu.

Pokud jde o volbu velikosti políčka, platí dvě protichůdné teze. Čím menší políčko zvolíme, tím je model a i výsledek celé simulace přesnější. Na druhou stranu čím menší plochu políčko

pokrývá, tím větší je velikost výsledného grafu (velikost roste dokonce s druhou mocninou) a to samozřejmě znatelně zpomalí vyhledávací algoritmus. Je tedy nutné najít potřebný kompromis.

3.2.2 Výhody rastrové reprezentace

Graf vytvořený z rastrové mapy má několik vlastností, které se dají s výhodou využít pro vyhledávání cesty a případnou optimalizaci. První z nich je znalost maximálního počtu uzlů, které jsou s aktuálním uzlem spojeny hranou. Z kapitoly 2.3.2 vyplývá, že maximální počet sousedů je 8. Je tedy zajištěno, že rozgenerování všech uzlů bude trvat přibližně stejnou dobu, že se tedy u žádného uzlu nijak výrazně dlouho vyhledávací algoritmus nezastaví a bude v grafu stále postupovat konstantní rychlostí dopředu.

Další nespornou výhodou je možnost snadného zachycení velké členitosti terénu a jeho různých změn. Tím, že každý uzel grafu reprezentuje jedno políčko mapy, které má přímo přiřazenou vlastní charakteristiku (výška, materiál), je popis celé mapy vždy jednotný, ať už mapa popisuje nějaké pohoří, nebo třeba rovnou homogenní plochu. Dá se tak snadno odhadnout časová náročnost vyhledávání jen podle rozměrů mapy a není nutné se zabývat detailnějšími informacemi, jak by tomu bylo u vektorového popisu.

3.2.3 Nevýhody rastrové reprezentace

Hlavní nevýhodou je nepřesnost popisu, která vzniká vždy, tvoříme-li rastrový popis mapy z jiné než z opět rastrové reprezentace. Každé políčko má přiřazen jistý přibližný údaj o části mapy kterou pokrývá, ale v případě, že ta není homogenní, musí logicky dojít ke ztrátě informací. Ta může být v jistých situacích významná, někdy dokonce kritická.

Představme si případ, že v mapě existuje zeď a v ní je průchod 2,5 m široký. Pokud by byla mapa pokryta sítí políček 1x1 m, pak by se velikost díry ve zdi jistě musela zaokrouhlit na celé číslo (například na 2 m). Pokud by se vyhledávala cesta pro objekt 2,2 m široký, pak by vyhledávací algoritmus průchod tímto místem nepovolil, i když by se do ní v reálu objekt pohodlně vešel. V druhém případě, byla-li by velikost průchodu zaokrouhlena na 3 m, pak by byla označena za průchozí i pro objekt 2,8 m široký a došlo by ke kolizi. Velikost chyby je tedy přímo úměrná hrubosti sítě, kterou mapu pokrýváme.

Druhou nevýhodou je velké množství generovaných uzlů i v případech, kdy to není nutné. Pokud by uprostřed celé mapy byla rovná plocha z jednoho materiálu o velikosti 100 m^2, a celá mapa by byla opět pokryta sítí políček 1x1 m, pak bude celá plocha pokryta 100 shodnými políčky (a tedy vyjádřena 100 uzly v grafu), i když by pro uložení stejné informace stačilo pouze jedno. Je zřejmé, že při rychlosti prohledávání grafu hraje významnou roli, prohledává-li se 100 uzlů, nebo jen uzel jediný.

3.2.4 Ohodnocení grafu

Pro praktické vytvoření kompletního grafu z rastrové reprezentace mapy musíme mimo vlastních uzlů určit existenci a hlavně ohodnocení všech hran. Způsob jakým se ohodnocení hran počítá je dopodrobna popsán v kapitole 2.5, nicméně ve vztahu (2.5) se vyskytuje konstanta *k*, jejíž hodnota není zatím přesně určena.

Konstanta *k* vyjadřuje míru obtížnosti průchodu mapou mezi dvěma místy. Její výpočet je závislý na mnoha aspektech. Měl by brát v úvahu typ terénu, výškový rozdíl daných míst, parametry objektu, který má daný úsek překonat a mnoho dalších.

Předpokládejme, že známe všechny relevantní údaje (aspekty) o terénu v obou daných místech mapy. Dále předpokládejme, že existuje míra ovlivnění objektu každým aspektem vyjádřená hodnotou z intervalu <0,1>. Nula znamená, že objekt není aspektem ovlivněn, jedna znamená maximální ovlivnění. Způsob výpočtu je potom následující:

1. Počáteční hodnotu *k* nastav na 1.
2. Zvol dosud neprozkoumaný aspekt *a*, pokud jsou již všechny aspekty prozkoumané vrať hodnotu *k*.
3. Vypočítej hodnotu aspektu beroucí v úvahu obě místa mapy (jako průměr, případně rozdíl hodnot).
4. Tuto hodnotu vyjádři jako desetinné číslo p_a v intervalu <0,1>.
5. Zjisti míru ovlivnění objektu daným aspektem o_a.
6. $k = k \cdot (1 + o_a \cdot p_a)$
7. Pokračuj bodem 2.

Výsledek, je vždy větší než jedna a vyjadřuje míru, kterou se podílejí všechny aspekty a parametry objektu na ohodnocení grafu. V praxi je nutné v algoritmu řešit individuální hodnoty některých aspektů a způsob jak se promítnou do závěrečného hodnocení (např. dokonale neprostupný terén), nicméně základní princip zůstává stejný. Pokud ovlivnění některým aspektem není lineární, ale kvadratické či jiné, není problém umocnit ve vzorci proměnnou p_a na příslušnou hodnotu, aniž by to změnilo jeho funkčnost.

3.3 Prohledávání grafu metodou A*

Pro algoritmus, který bude procházet graf a hledat v něm nejkratší cestu, jsem zvolil metodu A* (kapitola 2.3). Hlavní výhodou je, že tato metoda vybírá uzly prioritně podle předpokladu, že daný uzel leží na optimální cestě, a snaží se tedy postupovat co nejpříměji směrem k cílovému uzlu s ohledem na získané informace o mapě. Za tuto „vychytralost algoritmu“ se ovšem musí platit náročnější režií a dobou potřebnou pro prozkoumání jednotlivých uzlů. Tato negativa ovšem ve výsledku nepřeváží pozitiva a proto je A* vhodnou volbou pro nasazení v tomto projektu.

3.4 Abstraktní datové struktury pro A*

Při použití algoritmu A* (dle postupu z kapitoly 2.4.3) je nutné používat 2 seznamy (OPEN, CLOSE), přičemž po každém seznamu požadujeme specifické operace. Je proto vhodné použít pro implementaci daných seznamů takové abstraktní datové struktury (ADT), u kterých jsou právě tyto často prováděné operace co možná nejefektivnější.

3.4.1 Seznam OPEN

Nejčastější operace:

- vložení prvku
- nalezení a vyjmutí nejlépe ohodnoceného prvku
- test existence konkrétního prvku.

Časově nejnáročnější operací je (obvykle) nalezení nejlépe ohodnoceného prvku. Nabízí se použít například seřazené pole, nebo nějakou stromovou strukturu (binární strom) [IAL].

3.4.2 Seznam CLOSE

Nejčastější operace:

- vložení prvku
- test existence konkrétního prvku.

U seznamu CLOSE odpadá nutnost hledání prvku s nejnižším ohodnocením. Proto lze použít buď jednoduchý seznam, nebo pole, nebo ještě lépe strom, či hašovací tabulku [IAL].

3.4.3 Vlastní uzel

Pro každý uzel je vhodné, aby si udržoval informace o tom, jaké je jeho aktuální ohodnocení a také to, který uzel je jeho předchůdce (vhodné pro pozdější rekonstrukci cesty grafem). Proto je v projektu pro uložení uzlu použita jednoduchá struktura, která tato data obsahuje.

3.4.4 Použité ADT

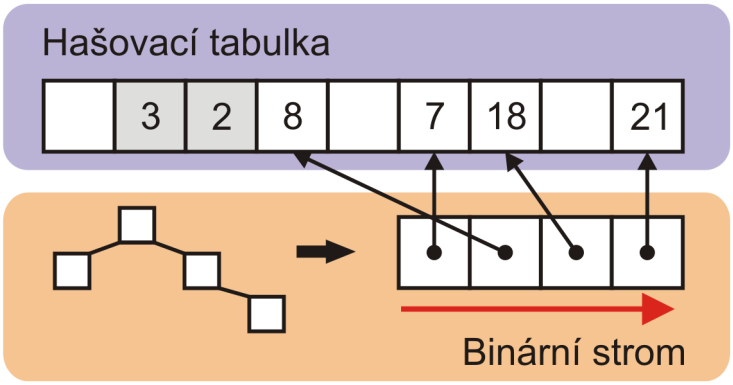
Vzhledem k tomu, že v průběhu algoritmu se uzly přesouvají mezi oběma seznamy a dvě často používané operace jsou pro oba seznamy stejné, zvolil jsem kompromisní řešení:
Existuje pouze jediný seznam, nad kterým se snadno provádí test existence konkrétního prvku. Tento seznam je reprezentován hašovací tabulkou. Jednotlivými prvky seznamu jsou struktury reprezentující uzly grafu, které obsahují tyto informace:

- příznak *open* (zda je uzel v OPEN či CLOSE)
- předchůdce (odkaz na svého předchůdce)

- ohodnocení
- vlastní data (identifikace uzlu v grafu)

Vkládání prvku i test na existenci jsou u hašovací tabulky potenciálně časově nenáročné operace (konstantní časová náročnost), nicméně tento údaj je závislý na efektivním využití tabulky a vhodné hašovací funkci.

Seznam OPEN neobsahuje uzly fyzicky, ale je tvořen pouze množinou odkazů do hašovací tabulky.



Obrázek 3.1: Vnitřní reprezentace OPEN a CLOSE

Tento seznam je uspořádán do binárního stromu. (S výhodou se dá využít standardní knihovny jazyka C++, kde je již strom implementován a není tedy nutné vytvářet vlastní implementaci a testovat její funkčnost).

Velkou nevýhodou použití hašovací tabulky je to, že se nedá dynamicky měnit její velikost. Je tedy třeba předem co nejlépe odhadnout celkový počet prvků, které v ní budou uloženy. Při nadhodnocení počtu prvků se zbytečně plýtvá pamětí, při podhodnocení se značně sníží rychlost operací nad tabulkou i při vhodně zvolené hašovací funkci. Je zřejmé, že pokud máme dostatek paměti, je lépe velikost tabulky udělat raději větší než menší. V praxi se velikost tabulky odvíjí od velikosti grafu a přímé vzdálenosti počátečního a koncového uzlu.

3.4.5 Časová složitost operací u použitých ADT

U klasického lineárně vázaného seznamu (nebo pole) o velikosti *N* jsou časové složitosti základních operací následující:

- Nalezení prvku: **O(N)** – projde všechny prvky
- Vložení prvku: **O(1)** – vloží na konec
- Test na existenci: **O(N)** – projde všechny prvky
- Výběr nejlepšího prvku: **O(N)** – projde všechny prvky
- Odstranění konkrétního prvku: **O(N)** – najde s **O(N)** a odstraní s **O(1)**

Je evidentní, že téměř u všech operací je časová složitost lineární a práce s takovýmito strukturami by byla značně pomalá. Existuje možnost využít i jiné ADT [AMIT], ale jako nejlepší pro použití modulu mi přišla hašovací tabulka a binární strom [IAL].

3.4.5.1 Hašovací tabulka

- Nalezení prvku: **O(1)** – v ideálním případě, pokud nedochází ke kolizím, v nejhorším případě **O(N).** V praxi se při dostatečné velikosti tabulky skutečná časová složitost blíží k ideálnímu případu.
- Vložení prvku – platí to samé jako při nalezení prvku.
- Test na existenci – platí to samé jako při nalezení prvku.

3.4.5.2 Binární strom:

- Vložení prvku: **O(log N)** - projití binárním stromem
- Výběr nejlepšího prvku: **O(log N)** - projití binárním stromem
- Test na existenci: **O(N)** – projde všechny prvky – obejde se přes hašovací tabulku

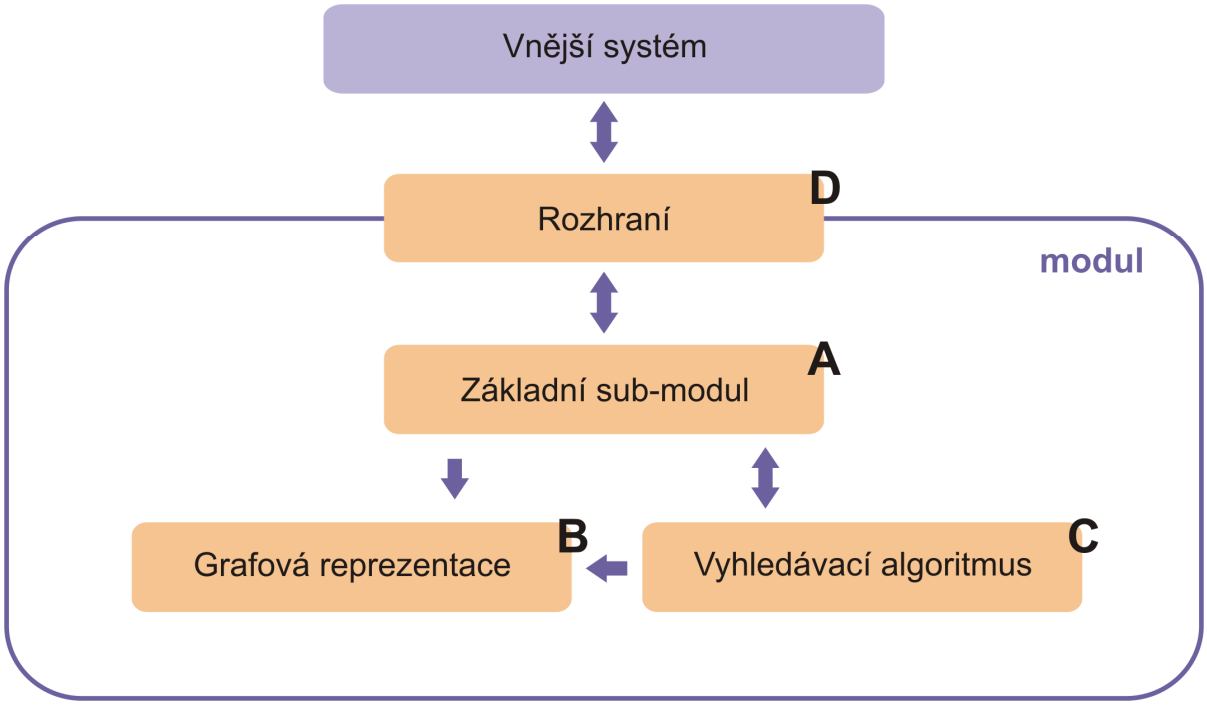
Všechny časové složitosti jsou velmi dobré. Nejhorší časová složitost je lineární a to pouze v případě, že je hašovací tabulka navržena nejhůře jak jen to je možné.

4 Výsledné řešení

Výsledné řešení je implementováno tak, aby bylo co nejvíce adaptabilní. To znamená, že například vyhledávací algoritmus pracující v modulu lze snadno zaměnit za jiný, aniž by se muselo zasahovat do vnitřní struktury celého modulu. Také se počítá s tím, že modul může být využit i v odlišných aplikacích než v té, pro kterou byl vyvíjen. Proto byl při vývoji kladen důraz na snadnou začlenitelnost do libovolného programu.

4.1 Rozvržení modulu

Celý modul je rozdělen do několika dílčích modulů (sub-modulů), jak je naznačeno na obrázku 4.1. Každý z nich plní specifickou funkci. Některý se stará o spravování mapy, jiný o komunikaci s vnějším systémem. Všechny sub-moduly jsou detailněji rozebrány v následujících kapitolách.



Obrázek 4.1: Základní rozdělení modulu

4.1.1 Základní sub-modul – A

Tvoří srdce celého modulu. Řídí práci ostatních sub-modulů a jejich vzájemnou komunikaci. Je stejně jako ostatní sub-moduly implementován jedinou třídou. V jejím konstruktoru se inicializují všechny potřebné komponenty. Pro vyhledání cesty se zavolá příslušná metoda této třídy s korektně nastavenými parametry.

4.1.2 Sub-modul pro správu ohodnoceného grafu – B

Tato část má na starost tvorbu a správu grafové reprezentace se kterou celý modul pracuje. Je také implantována zvláštní třídou. Ta při své inicializaci komunikuje s rozhraním D, přes které jsou jí z venkovní aplikace poskytnuty všechny informace o mapě. Získání dat a jejich vlastní uložení je podrobně popsáno v kapitole 3.2.

V průběhu vlastního vyhledávání se volá metoda pro získání informací o konkrétním uzlu mapy. Tato metoda očekává jako parametr jednoznačný identifikátor uzlu v grafu (souřadnice, vrstva) a vrací všechny uzly, ke kterým z daného uzlu vede hrana, společně s jejich ohodnocením.

4.1.3 Sub-modul pro prohledávání grafu – C

V tomto sub-modulu probíhá vlastní vyhledávání. V obecné rovině je tento modul tvořen abstraktní třídou, ze které lze pomocí dědičnosti vytvořit konkrétní implementaci. Ve výsledném řešení odpovídá konkrétní implementace návrhu z kapitoly 3.3 (algoritmus A*), nicméně celou třídu lze implementovat i jinak (použit odlišnou vyhledávací metodu, nebo jiné datové struktury).

Sub-modulu jsou nejdříve předány informace o počátečním a cílovém uzlu. Sám pak dočasně přebírá řízení celého vyhledávání, prozkoumává graf a hledá požadovanou cestu. Informace o úspěchu či neúspěchu nakonec předává zpět řídícímu prvku.

4.1.4 Sub-modul rozhraní – D

Tato část modulu zajišťuje předání informací o mapě z aplikace do modulu a není jako jediná ze zřejmých důvodů v modulu implementována. Existuje pouze abstraktní třída tvořící rozhraní tohoto sub-modulu. Konkrétní implementace je zcela závislá na aplikaci, ve které se modul použije.

Všechny metody této třídy jsou volány při inicializaci celého modulu. Jejich využití v průběhu dalšího života modulu má význam pouze v případech, kdy se celá mapa dynamicky mění.

Obsahuje dvě metody pro získání základních rozměrů mapy a dále metody, které se dotazují na informace o konkrétním místě na mapě. Správná implementace všech těchto metod je pro celkovou funkčnost modulu naprosto nezbytná.

4.2 Životní cyklus vlastního modulu

Životní cyklus modulu lze popsat ve dvou krocích, kde se druhá část může libovolně opakovat:

1. Inicializace vyhledávacího modulu. Při ní se inicializují všechny dílčí moduly a vytvoří se grafová reprezentace původní mapy. Řízení je předáváno sub-modulům v tomto pořadí: A, C, A, B
2. Vyhledání konkrétní cesty. Předají se údaje o počátečním a cílovém uzlu do části mající na starost vyhledávání. Ta již sama komunikuje s grafovou reprezentací a pokusí se cestu

vyhledat. Při úspěchu se cesta vrací zpět do stávající aplikace. Řízení je předáváno sub-modulům v tomto pořadí: A, C, B, C, B, … C, B, A.

4.3 Použití modulu

Při praktickém nasazení a používání vyhledávacího modulu je nezbytné učinit dva kroky.

V první řadě to je implementace třídy rozhraní. Vzhledem k možnosti různých typů a velikostí map je nejprve nutné zvolit velikost základní mřížky, kterou bude mapa pokryta. Pro každé políčko mřížky musí být systém schopný vrátit sumarizované údaje o ploše, kterou pokrývá. Náznak řešení tohoto úkolu je blíže popsán v kapitole 3.2.1.

Teď je již modul schopen vytvořit mapu a je připraven k vyhledávání. Nicméně vyvstává druhý problém. Jak modulu sdělit všechny podrobné nastavení vyhledávání? Toto zajišťuje speciální třída, jejíž instance se očekává jako vstupní parametr metody, která zajišťuje vyhledávání. Objekt této třídy je tedy nutné před samotným vyhledáním inicializovat na příslušné hodnoty a až potom modul žádat o samotné nalezení cesty.

Pokud je vše správně nastaveno, modul najde v dané mapě nejlepší cestu podle zadaných kritérií.

5 Optimalizace

Výše popsané řešení splňuje základní podmínky kladené na celý projekt. Lze vyhledat cestu v dané mapě a pokud taková cesta existuje a zároveň se nepříznivě neprojeví zaokrouhlovací chyby při tvorbě mapy, pak bude nalezená cesta optimální. Nicméně vyhledávání by bylo u velkých map vzhledem k rozsáhlosti prohledávaného grafu extrémně pomalé.

Celý modul by se tak stal pro praktické využití téměř nepoužitelným, i přes to, že by vlastně dělal všechno správně. V aplikacích, kde je nutné najít cestu v určitém čase, i třeba za cenu horšího řešení, by se v takovéto podobě nedal použít. Je tedy nutné mít možnost vyhledávat rychleji i na úkor kvality.

5.1 Velikost grafu

Rychlost vyhledávacích algoritmů nejvíce záleží na velikosti stavového prostoru, který musejí prozkoumat [IZU]. Bylo by dobré mít možnost určit, jak maximálně velký stavový prostor může být, aby se dala odhadnout nejzazší doba, za kterou bude řešení známo.

Nejjednodušší způsob, jak tohoto dosáhnout, je prohlásit nějaký konkrétní rozměr mapy za maximální a pro větší mapu prohledávání nepovolit. Nicméně by se tím opět omezila použitelnost modulu.

Lepším řešením je nějakým způsobem zmenšit velikost stavového prostoru (tedy snížit počet uzlů prohledávaného grafu).

Toho lze docílit například projitím celého grafu a slučováním sousedních uzlů se stejnými vlastnostmi tak dlouho, dokud to lze. Tím se velikost grafu téměř vždy zredukuje, nicméně se tím ztratí pravidelnost rozčlenění mapy a tím i možnost jednoduchého uložení dat v počítači. Navíc neexistuje jistota, že se graf podaří zredukovat natolik, aby se prohledávání zrychlilo na požadovanou úroveň.

Další možností je postavit nad základním grafem nový graf, který by byl jakousi přibližnou kopií původního grafu, zachoval by si pravidelnost, ale obsahoval by menší počet uzlů. Takový graf vznikne například tak, že sloučíme vždy 4 uzly (reprezentující políčka poskládaná do čtverce) do jednoho většího, který bude tyto uzly zastupovat. Popis nového uzlu se získá vhodným skloubením údajů z původních uzlů (průměr, maximální hodnota, atd.). Nový graf má tedy 4x méně uzlů, což je znatelné zmenšení, nicméně vždy je možné pracovat se 4x větší mapou, takže problém se tím nevyřeší, ale pouze oddálí. Není ovšem problém celou proceduru vytvoření redukovaného grafu opakovat. To lze udělat snadno tak, že nový graf prohlásíme za graf původní a zbytek postupu je totožný. Tímto způsobem můžeme postupně graf redukovat až do doby, kdy bude obsahovat pouze

jediný uzel. Existuje tedy možnost převést jakýkoliv graf na graf s počtem uzlů menším, nebo rovným libovolnému přirozenému číslu.

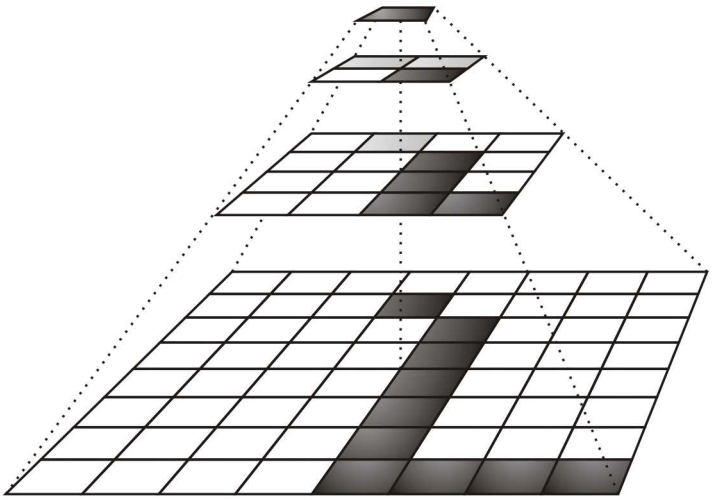
5.1.1 Stromové uložení

Vytváříme-li redukované grafy postupem z předchozí kapitoly, můžeme získat několik různých reprezentací téhož grafu, lišící se v množství uzlů. Počet všech takovýchto reprezentací označme jako *n*. Každý nový graf má přibližně 4x méně uzlů, než graf předchozí (je třeba řešit situace, kdy velikost mapy v některém směru není dělitelná dvěma). Označíme-li počet uzlů původního grafu jako *p*, platí vztah (5.1):

$$p \leq 4^{n},\ n \approx \log_4 p \tag{5.1}$$

Je výhodné mít možnost kdykoliv pracovat s kteroukoliv reprezentací mapy, případně mít možnost plynule přejít k práci s jemnější či hrubší reprezentací. Pro snadný přechod mezi grafy a pro snadné vyhledávání konkrétního uzlu v daném grafu je vhodné uložit grafy do struktury, která je pro tyto operace vhodná. Vzhledem k operacím a zmenšujícímu se počtu uzlů u každého nového grafu až do grafu reprezentovaného pouze jediným uzlem se nabízí pro uložení všech grafů stromová struktura [IAL].

Celý strom tedy vypadá následovně. Graf reprezentovaný jediným uzlem nazvěme kořenem stromu. Za vrstvu stromu označme ty uzly, které mají od kořene stejnou vzdálenost. V každé vrstvě stromu jsou uloženy uzly patřící ke stejné reprezentaci grafu. V nejnižší vrstvě stromu je vlastně uložen po uzlech původní graf. Každý uzel grafu obsahuje odkaz na svojí redukovanou podobu (strom je vytvářen zdola nahoru). Ke konkrétnímu uzlu v dané vrstvě přistoupíme tak, že najdeme uzel originálního stromu a přes ukazatele se dostaneme až na hledaný uzel.



Obrázek 5.1: Stromové uložení různých grafových reprezentací

Označíme-li graf uložený ve vrstvě *n* jako g_n a maximální počet uzlů takového grafu jako $c(g_n)$ pak platí vztah (5.2)

$$c(g_{n+1}) = c(g_n)/4 \tag{5.2}$$

5.2 Prohledávání ve vrstvách

Existuje-li několik grafových reprezentací mapy, pak si může vyhledávací algoritmus vybrat, ve které z nich bude vyhledávat. Je zřejmé, že čím nižší vrstva to bude, tím může být nalezené řešení přesnější, ale také tím déle může jeho nalezení trvat.

Díky možnosti vyhledávání ve vrstvách se otevírá celá řada nových možností, jak vyhledávací proces upravit podle různých kritérií. Například lze snadno a rychle vytvořit odhad správné cesty (vyhledáním celé cesty v některé z vyšších vrstev stromu) a z tohoto odhadu zjemnit například prvních 20% nalezené cesty na úroveň nejnižší reprezentace. V případě potřeby se pak dopočítá další část cesty. To lze s výhodou využít v situacích, kdy není dostatek času pro vyhledávání celé cesty, nebo v případech, kdy objekt často mění cíl své cesty, aniž by dorazil do cíle. V těchto případech se tedy nemusí provádět zbytečně celý zdlouhavý výpočet. Samozřejmě, že nalezená cesta (ani její část) nemusí být optimální, jelikož dočasný cílový bod (v příkladu v jedné pětině cesty) byl vypočítán také pouze přibližně, a optimální cesta přes něj vůbec nemusí vést.

Další způsob využití grafové reprezentace mapy je možnost dynamické změny vrstev během prohledávání. Uzel v každé vrstvě může nést informaci o tom, jakou chybu obsahují jeho vlastní data a v případě, že by tato chyba byla větší, než jistá zanedbatelná mez, pak se dynamicky přejde do vrstvy nižší. Naopak v případě, že se prohledává velká celistvá plocha, by bylo zbytečné vyhledávat v nejnižší vrstvě, když stejnou informaci může poskytnout vrstva vyšší, a lze se v ní navíc pohybovat po větších skocích. Toto vylepšení by umožňovalo zrychlit, nebo zpřesnit celé prohledávání v závislosti na povrchu mapy, nad kterou se vyhledávací algoritmus zrovna pohybuje, nicméně není ve výsledné verzi modulu implementováno.

5.3 Uložení do sektorů

Vylepšením popsaným výše se sice zrychlilo vyhledávání cest, ovšem cenou za to jsou větší nároky na množství paměti spotřebované pro uložení celého stromu se všemi dostupnými grafovými reprezentacemi. U velkých map a rozsáhlých aplikací, do kterých bude modul implementován, může být problém mít všechna data neustále uložena v paměti. Proto je v modulu možnost odkládat si data některá na disk.

Místa, mezi kterými se cesta vyhledává, mohou ležet na opačných koncích mapy, ale obvykle tomu tak nebývá. Často je nutné posunout objekt pouze o nějakou kratší vzdálenost, nebo například drobně poupravit jeho pozici vzhledem k pozici nepřítele. Proto není nutné mít v paměti nahranou celou mapu, ale pouze tu část, ve které se s největší pravděpodobností bude vyhledávat. V modulu se to řeší pomocí sektorů. Celá mapa je rozdělena na několik sektorů. Jejich přesný počet závisí na velikosti celé mapy, ovšem je to vždy násobek čísla 4. Sektor je oblast mapy (grafu), která vznikne tak, že zvolíme jednu vrstvu, a každý uzel v ní ležící označíme jako kořen daného sektoru. Tím nám

vznikne tolik sektorů, kolik je uzlů v dané vrstvě, a každý sektor bude pokrývat čtvercovou oblast určenou svým kořenovým uzlem. Každý sektor lze kdykoliv odložit z paměti do pomocného souboru na disk a v případě potřeby jej zase v nezměněné podobě načíst.

Celá práce se sektory tedy probíhá tak, že před zahájením hledání jsou všechny sektory uložené na disku. Načte se sektor obsahující počáteční uzel a začne se vyhledávat cesta. V momentě, kdy algoritmus dojde do místa, které leží v sektoru, jenž není uložen v paměti, načte se tento sektor a vyhledávání pokračuje. Existuje možnost odložit sektor zpět na disk i v průběhu vyhledávání v případě, že jsou již všechny uzly daného sektoru prozkoumané, nebo nedostupné.

6 Testování

Praktická funkčnost celého modulu byla testována ve zvláštní pro tento účel navržené aplikaci, která je součástí přílohy. Není v ní implementován žádný simulační systém, dokáže pouze vytvořit digitální mapu z RGB hodnot obrázku a tuto mapu společně s údaji o vyhledávané cestě potom poskytnout vyhledávacímu modulu. Každému políčku mapy jsou přiřazeny tři základní údaje –výška, prostupnost terénu a ohodnocení bezpečnosti průchodu daným políčkem. Detailnější popis aplikace je v příloze.

V této aplikaci jsem provedl sadu testů, ve kterých jsem hodnotil přesnost nalezené cesty a čas, který vyhledávání, v závislosti na velikosti mapy a na jemnosti grafu (vrstvě), který byl pro vyhledávání použit. Cesta byla vždy vyhledávána mezi protilehlými body mapy.

Nejmenší vzdálenost mezi sousedními body výsledného vektoru nezávisí na vrstvě ve které se vyhledává. Je to dáno tím, že cesta nalezená ve vyšší než základní vrstvě se dále zjemňuje dílčím vyhledáváním v nižších vrstvách grafu. Z toho důvodu není rychlost výpočtu přímo úměrná vrstvě, ve které se vyhledává.

Ve výsledcích jsou uvedeny tyto základní údaje:

- Velikost mapy
- Vrstva – jak hrubý je graf, ve kterém se provede základní vyhledání
- Čas – přibližný čas potřebný pro nalezení cesty (informace o čase mají význam pouze při vzájemném porovnávání)
- Odchylka – údaj o kolik procent se ohodnocení výsledné cesty lišilo od optimální cesty

6.1 Homogenní mapa

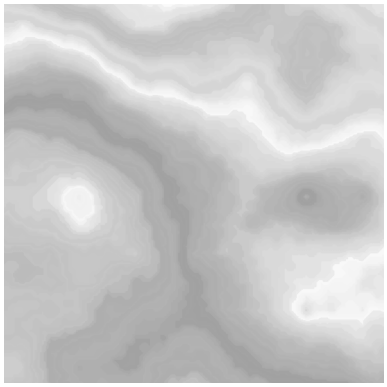
V tomto testu se cesta vyhledávala na mapě se všemi políčky shodnými.

VELIKOST MAPY (š x v)	VRSTVA	ČAS	ODCHYLKA
1024x1024	1	0 s	0,0 %
1024x1024	2	2 s	0,0 %
1024x1024	3	1 s	0,0 %
1024x1024	4	2 s	0,0 %
1024x1024	5	2 s	0,0 %

Vyhledávání ve vyšších vrstvách zabralo více času. Je to způsobeno tím, že nalezená cesta se dále zjemňuje až na úroveň nejnižší vrstvy.

6.2 Výškově členitá mapa

V tomto testu se políčka vzájemně liší pouze nadmořskou výškou. Reliéf mapy je na obrázku 6.1. V této mapě by se již měla projevit vyšší rychlost při vyhledávání ve vyšších vrstvách stromového uložení grafu.



Obrázek 6.1: Výškový reliéf mapy

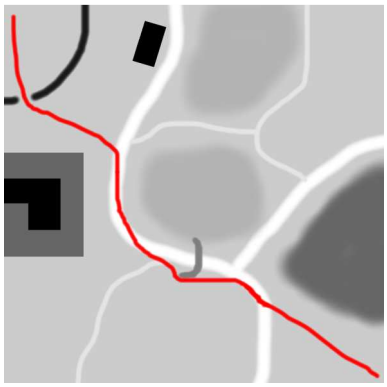
VELIKOST MAPY (š x v)	VRSTVA	ČAS	ODCHYLKA
1024x1024	1	18 s	0,0 %
1024x1024	2	6 s	0,45 %
1024x1024	3	2 s	0,29 %
1024x1024	4	3 s	0,70 %
1024x1024	5	2 s	1.42 %
1024x1024	6	1 s	0,52 %

Nejlepšího výsledku (poměr přesnost / čas) bylo dosaženo při prohledávání ve třetí vrstvě. Je zajímavé, že vyhledání v šesté vrstvě je v některých případech přesnější, než ve vrstvách nižších, ovšem neplatí to vždy.

6.3 Obecná mapa

Poslední typ testů byl proveden s mapou, která se snaží alespoň přibližně zachytit skutečné prostředí. Obsahuje různé typy terénu, je výškově členitá a je určena také různá míra nebezpečí průchodu pro každé políčko. Tato mapa byla otestována v různém rozlišení. Detaily mapy jsou na obrázcích 6.2, 6.3 a 6.4.

VELIKOST MAPY (š x v)	VRSTVA	ČAS	ODCHYLKA
1024x1024	1	63 s	0,0 %
1024x1024	2	18 s	0,26 %
1024x1024	3	7 s	0,55 %
1024x1024	4	6 s	5,7 %
1024x1024	5	6 s	7,5 %
1024x1024	6	6 s	6,7 %
2048x2048	1	751 s	0.0 %
2048x2048	3	55 s	1.1 %
2048x2048	5	19 s	2.5 %
2048x2048	6	10 s	4.3 %
256x256	1	2 s	0.0 %
256x256	2	1 s	0.52 %
256x256	3	1 s	5.9 %
256x256	6	2 s	5.8 %



Obrázek 6.2: Terénní členitost společně s optimální cestou

Obrázek 6.3: Výšková členitost

Obrázek 6.4: Členitost nebezpečí

Testy ukazují, že při volbě vhodné prohledávací vrstvy záleží především na velikosti a členitosti mapy. Nelze určit jedno nastavení jako nejlepší a používat jej na všechny typy map. Z testů plyne důležitý závěr, že prohledávat obrovské mapy bez použití vyšších vrstev by práci celého modulu naprosto znehodnotilo.

7 Závěr

Při návrhu a implementaci modulu byly splněny všechny části zadání. Za předpokladu, že je modul do aplikace správně integrován, poskytuje možnost vyhledávat cestu v daném terénu pro libovolný objekt podle zadaných kritérií. Umí jak vyhledat optimální cestu, tak nalézt alespoň přibližnou cestu (nebo její část) v jistém časovém úseku. Tato vlastnost může být s výhodou využita především u simulací běžících v reálném čase, kdy systém nemůže věnovat vyhledávání dostatek zdrojů po dostatečně dlouhou dobu, ale je třeba najít alespoň přibližnou cestu „okamžitě“. Modul byl otestován na speciální aplikaci s různými typy map s uspokojivými výsledky. Ostré nasazení do skutečného simulačního systému a tedy jeho skutečné prověření jej však teprve čeká a zajisté se to neobejde bez drobných úprav. Ty ovšem přesahují rámec tohoto projektu.

Na vývoji a vylepšeních modulu by se dalo ještě dlouze pracovat a je to jedna z možností, kam by se mohla ubírat moje diplomová práce. Modul by se mohl rozšířit co se týče univerzality, mohl by umět pracovat s různými typy map, případně by mohl dokázat vyhledávat ve 3D prostoru. Mimo nové funkcionality by se dalo pracovat na dalších optimalizacích stávajícího řešení. Modul by mohl pracovat s breakpointy, předpočítávat si výsledky nějakých často vyhledávaných cest, atd.

Celý modul byl vyvíjen s ohledem na praktické využití v simulačním systému firmy, která projekt podporovala. Proto byla všechna rozhodnutí, volící mezi různými způsoby řešení nějakého dílčího problému, podřízena co nejlepší funkčnosti pro daný firemní systém. Ten je zaměřen na vojenské simulace ve členitých venkovních mapách, to znamená, že modul dosahuje nejlepších výsledků právě s takovým typem map. Nicméně lze jej nasadit obecně na libovolné typy terénů. Případně se vzhledem k modulárnímu návrhu dá snadno přepracovat pouze některá jeho nevyhovující část.

Práce na projektu mě obohatila o mnoho nových zkušeností. Rozšířil jsem si obzory v oblasti prohledávacích algoritmů a umělé inteligence obecně. Především mi ale poskytla možnost vymyslet a aplikovat své vlastní způsoby řešení některých dílčích problémů. U vlastních nápadů bych se rád krátce zastavil.

V první řadě jsem se snažil o zvýšení rychlosti prohledávacího algoritmu vhodnou volbou datových struktur. Kombinace hašovací tabulky a binárního stromu má v porovnání s ostatními strukturami velmi dobré vlastnosti (především nízká časová složitost), které jsou pro rychlé nalezení cesty klíčové.

Algoritmus ohodnocení cesty mezi dvěma uzly grafu popsaný v kapitole 3.2.4 je také původní, ale zda se osvědčí ukáže až nasazení modulu do reálné aplikace.

Poslední nápad, který zde zmíním, je pokus o nalezení způsobu, jakým by bylo možné urychlit prohledávání velkých map, třeba i za cenu ztráty optimálního řešení. Nakonec jsem toho dosáhl

tvorbou stromové struktury nad celou mapou a možností vyhledávat v různých vrstvách stromu, díky čemuž se vyhledávací algoritmus v mapě pohybuje po větších skocích a je tak rychlejší.

Literatura

[AMIT] Patel, A.: Game Programming, Dokument dostupný na URL http://theory.stanford.edu/~amitp/GameProgramming/ (duben 2007).

[AN] Norbert Arvid: Time-sliced pathfinding on arbitrary polygon surfaces, Dokument dostupný na URL www.cs.umu.se/education/examina/Rapporter/ArvidNorberg.pdf (duben 2007).

[AH] Hildebrand, A.: Referát do předmětu Praktikum z informatiky, Dokument dostupný na URL http://pathlib.hildebrand.cz/doc/Referat/pathref.html (duben 2007).

[FMJ] Jönsson, M. F.: An optimal pathfinder for vehicles in real-world digital terrain maps, Dokument dostupný na URL http://www.student.nada.kth.se/~f93-maj/pathfinder/ (duben 2007).

[IZU] Zbořil, F.: Studijní opora předmětu IZU, VUT v Brně, Fakulta informačních technologií, ÚITS, 2006.

[IAL] Honzík, J. M.: Studijní opora předmětu IAL, VUT v Brně, Fakulta informačních technologií, ÚITS, 2006.

[WIKI] Wikipedia, Dokument dostupný na URL http://cs.wikipedia.org (duben 2007).

Seznam příloh

Obsah CD

Na přiloženém CD se nachází:

- Zdrojový kód modulu (`source/`)
- Funkční testovací aplikace se zdrojovými kódy (`test/`)
- Programová dokumentace (`doc/`)
- Elektronická podoba textové zprávy (`xvanaj00.pdf`)

Praktické nasazení vyhledávacího modulu

V této příloze je popsán způsob nasazení modulu útržky kódu v jazyce C++. Před praktickým nasazením do reálného prostředí doporučuji pročíst si programovou dokumentaci a podívat se na třídy definované v hlavičkovém souboru `interface.h`. Znalost těchto tříd je totiž pro používání modulu nezbytná. Teď již k vlastnímu postupu.

- Předpokládejme, že celý modul je v adresáři `./module`. Nejprve vložíme do programu hlavičkový soubor modulu.

```
#include "module/path_finder.h"
```

- Vytvoříme vlastní implementaci třídy rozhraní (`T_interface`).

```
// vlastní třída implementující rozhraní pro práci s mapou dle prostředí
class T_my_map_interface : public T_interface {
   public:
      // konstruktor
      T_my_map_interface(){
         //...
      }
      // vlastní implementace všech abstraktních metod
      //vrací x-ový rozměr mapy
      virtual int get_x(){
         // ...
      }
      // ...
};
```

- Celý modul začleníme do systému takto:

```
// začlenění modulu

// instance rozhraní
T_my_map_interface rozhrani;

// instance vyhledávacího modulu
T_path_finder pathfinder(rozhrani);
```

- Vlastní použití modulu potom vypadá následovně (pro ukázku hledáme cestu mezi body [5,10] a [15,60] pro objekt 5 jednotek široký a stačí nám vypočítat jen prvních 25% cesty).

```
// inicializace paramtrů
T_params p(5, 10, 15, 16);  // odkud, kam
p.relative(25);             // typ výpočtu cesty
p.set_width(5.0);           // šířka objektu

//vlastní vyhledání
if (pathfinder.find(p)){
   vector <point> * cesta = pathfinder.get_way();
   //... zpracování cesty ...
}
```

Nasazení celého modulu není složitou záležitostí, problémem může být pouze vytvoření vlastní implementace rozhraní a správná inicializace parametrů. Praktický příklad obojího lze nastudovat ve zdrojových kódech testovací aplikace.

Testovací aplikace

K testování funkčnosti celého modulu byla vytvořena speciální aplikace v jazyce C++. Kvůli lepší vizualizaci výsledků používá tato aplikace grafické uživatelské rozhraní (glut - OpenGL).

Celý program umožňuje testovat modul tím způsobem, že z obrázku v grafickém formátu BMP vytvoří rastrovou mapu. Z barevných složek každého pixelu určí vlastnost daného políčka. Z červené nadmořskou výšku, ze zelené propustnost terénu a z modré míru nebezpečí průchodu políčka (využití například při průchodu nepřátelským územím). Takto vytvořenou mapu předá vyhledávacímu modulu.

Spustitelný soubor se jmenuje `sim.exe` a při spuštění očekává jediný parametr, kterým je cesta k obrázku ve formátu BMP, ze kterého se mapa vytváří.

Po spuštění se objeví okno, ve kterém je zobrazena daná mapa. Na mapě je modrým bodem znázorněno výchozí místo a červeným bodem místo cílové. Po úspěšném vyhledávání cesty se její tvar promítne na povrch mapy formou žluté lomené čáry.

Aplikace je přenositelná mezi různými platformami za předpokladu existence potřebných knihoven. Byla otestována na operačních systémech Windows XP a Fedora 5.0.

Ovládání

Program lze ovládat pomocí myši a klávesnice.

- `držení levého či pravého tlačítka myši` - změna pohledu kamery
- `kurzorové šipky` – pohyb počátečním bodem
- `SHIFT` + `kurzorové šipky` – pohyb koncovým bodem
- `ENTER` – zahájení vyhledávání
- `0-9` – výběr vrstvy, ve které se bude prohledávat (0 - nejnižší)
- `ESC` – konec aplikace

Obrázky z programu