VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

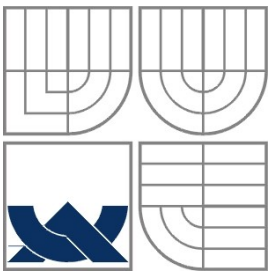
DEMONSTRACE METOD PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

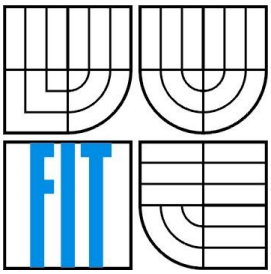
AUTOR PRÁCE
AUTHOR

MARTIN TUREČEK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

DEMONSTRACE METOD PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU

DEMONSTRATION OF STATE SPACE SEARCH METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN TUREČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. FRANTIŠEK ZBOŘIL, Ph.D.

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Tureček Martin**
Obor: Informační technologie
Téma: **Demonstrace metod prohledávání stavového prostoru**
Kategorie: Umělá inteligence

Pokyny:

1. Navrhněte prostředí pro demonstraci principů fungování algoritmlů pro prohledávání stavového prostoru.
2. Zvolte několik úloh vhodných pro ukázání fungování jak informovaných, tak neinformovaných metod.
3. Implementujte demonstraci tak, aby bylo možné experimentovat s volbami heuristik a zobrazovat nejen výsledná řešení a jejich složitosti, ale i průběhy hledání řešení.
4. Jako součást práce vytvořte manuál pro danou aplikaci a popis demonstrovaných metod ve formě webových stránek.

Literatura:

- Russel, P., Norvig, S.: Artificial Intelligence - A Modern Approach, Prentice Hall, 2002

Při obhajobě semestrální části projektu je požadováno:

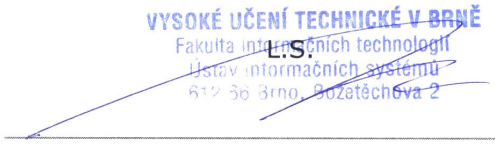
- první dva body zadání

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 nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zbořil František, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008



L.S.

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: **Martin Tureček**
Id studenta: 79283
Bytem: Nerudova 252, 697 01 Kyjov
Narozen: 23. 07. 1986, Kyjov
(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: Demonstrace metod prohledávání stavového prostoru
Vedoucí/školitel VŠKP: Zbořil František, 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

Tato bakalářská práce popisuje problematiku metod prohledávání stavového prostoru. Práce se zaměřila zejména na praktickou část, jejímž cílem bylo vytvořit demonstrační aplikaci, která by měla sloužit jako pomůcka do předmětu Základy umělé inteligence. Výsledná aplikace tento požadavek splňuje zejména díky jednoduchému ovládání, možnosti krokovat algoritmus a hlavně přitažlivým grafickým kabátkem. Aplikace je navíc implementována jako javovský applet, takže by její spuštění mělo být možné odkudkoliv.

Klíčová slova

informované metody, stavový prostor, A* algoritmus, java, heuristika, vyhledávání cesty

Abstract

This bachelor's thesis describes the issue of state space search methods. Thesis was focused namely on the practical part. Its main goal was to create a demonstration application which should serve as an additional help for Fundamentals of Artificial Intelligence course. The resulting application satisfies this requirement thanks to an easy control, a possibility of steping through the alghoritm and an attractive graphic layout. The application is also implemented as a java applet, so its start should be possible from anywhere.

Keywords

inform methods, state space, A* alghoritm, java, heuristic, pathfinding

Citace

Martin Tureček: Demonstrace metod prohledávání stavového prostoru, bakalářská práce, Brno, FIT VUT v Brně, 2008

Demonstrace metod prohledávání stavového prostoru

Prohlášení

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

……
Martin Tureček
14.5.2008

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Františku Zbořilovi Ph.D., který mi byl po celou dobu tvorby tohoto projektu vždy ochotně nápomocen.
Dále bych rád poděkoval Františku Smrčkovi, který zachránil aplikaci před mým grafickým antitaletnem a vytvořil pro ni velmi hezké grafické znázornění terénu.

Obsah

1 Úvod

V dnešní době přenáší lidstvo čím dál tím více zodpovědnosti na stroje, většinou řízené počítači. Snaží se tím minimalizovat nehody, které se stávají díky tomu, že člověk je obecně tvor často chybující. Tento fakt ovšem klade znovu nároky na člověka jako takového, protože je to on, kdo stroje naprogramuje tak, aby dělaly, co od nich požaduje. Oboru, který se touto problematikou zabývá, říkáme *umělá inteligence* a využití jeho poznatků je v dnešní době velmi rozšířené.

Je tomu už dávno, co jsem do jedné hry potřeboval implementovat algoritmus, který by pomohl mému pixelovému hrdinovi překonat strastiplnou cestu po herní mapě. Narazil jsem sice na spoustu teoretických rozborů dané problematiky, ale demonstračních aplikací jsem našel velmi poskrovnu a na kvalitní jsem nenarazil vůbec. Proto vznikla myšlenka právě takovou aplikaci stvořit, aby programátorům usnadnila jejich začátky.

Tato práce se tedy bude zabývat metodami, kterými je výše popsaný problém možno řešit, zejména se pak zaměří na ty informované, hlavně pak královnu této kategorie, metodu A*. První kapitola této práce bude popisovat dostupné metody, zaměří se na jejich porovnání a použitelnost a také si zde vysvětlíme, proč pro danou úlohu použijeme právě metodu A*, kterou detailněji popíšeme v druhé kapitole.

Třetí kapitolou se konečně dostaneme k samotnému návrhu aplikace, povíme si, jaké metody použijeme, aby aplikace byla pro studenta dané problematiky něčím zajímavá a tím pádem použitelná jako didaktická pomůcka.

V pořadí čtvrtá kapitola bude pojednávat o samotné implementaci aplikace. Zdůvodní volbu programovacího jazyka a nástrojů, poté popíše základní strukturu aplikace a nakonec postup při jejím programování.

V poslední kapitole zhodnotíme výsledky naší práce, posoudíme její přínos a v neposlední řadě se zamyslíme nad tím, jak bychom ji mohli do budoucna vylepšit.

Na CD jsou k připojeny zdrojové soubory aplikace, programátorská dokumentace a elektronická verze textu, který právě čtete.

2 Metody prohledávání stavového prostoru

Tyto metody jsou jednou z kategorií, které řadíme do podkapitoly řešení úloh v oblasti umělé inteligence. Nejdůležitějším pojmem je právě *stavový prostor*, který lze definovat jako dvojici *(S, O)*. S je množina všech stavů, do kterých se může úloha dostat. *O* je pak množina všech operátorů, které jsou schopny stavy úlohy měnit. Každá úloha je pak definována jako dvojice (s_0, G), kde s_0 je počáteční stav úlohy, přičemž tento prvek je prvkem výše zmíněné množiny *S*. *G* je pak množina všech cílových stavů dané úlohy a opět je nutno zmínit, že se jedná o podmnožinu množiny *S*. Existuje spousta metod, které jsou danou úlohu schopny řešit a jejich výsledkem je vždy posloupnost operátorů z množiny *O*, které transformují úlohu ze stavu s_0 do jednoho ze stavů množiny *G*. Jejich základní odlišnost je v tom, zda mají nějakou informaci o stavu cílovém, díky které pak jsou schopny řešení zefektivnit. Podle této vlastnosti se pak tyto metody dělí na informované a neinformované. Metody při prohledávání stavového prostoru vytvářejí strom, který může být k naší výhodě použit velmi efektivně pro demonstraci toho, jak daný algoritmus funguje. Uzly tohoto stromu reprezentují stav úlohy a hrany mezi nimi pak operátory, které jsme mohly na daný stav aplikovat a transformovaly tak úlohu do jiného stavu.

2.1 Neinformované metody

Jak bylo výše řečeno, tyto metody nemají byť jedinou informaci o cílovém stavu, ani žádné prostředky, jak aktuální stavy hodnotit. Ačkoliv jsou tyto metody daleko pomalejší, než metody informované, často nám, stejně jako v reálném životě, nezbyde nic jiného, než po nich sáhnout. Popíšu zde pár příkladů, ale rozhodně není účelem a smyslem této dokumentace zacházet do nějakého detailního popisu, či snad vyjmenovávání všech metod.

2.1.1 Prohledávání do šířky

Jedná se o nejzákladnější neinformovanou metodu, která se hodí víceméně jen na jednodušší úlohy. Nicméně právě díky tomu je tato metoda ideální jako úvod do problematiky. Poprvé se zde také vyskytuje fronta OPEN, která se bude vyskytovat napříč všemi metodami, ovšem jako jiná datová struktura (např. seznam nebo zásobník). Fronta OPEN slouží k tomu, aby si do ní algoritmus odkládal všechny myslitelné následníky právě expandovaného uzlu. V každém kroku si pak z této fronty

vybere jeden prvek, který může dále expandovat. Slovně by se tento algoritmus dal popsat následovně:

1. Sestroj frontu OPEN a přidej do ní počáteční uzel
2. Vyber z fronty OPEN první uzel, pokud ovšem ve frontě OPEN není žádný prvek, tak úloha očividně nemá řešení
3. Je-li vybraný uzel cílový, pak ukonči algoritmus s tím, že právě vybraný uzel je cílovým stavem dané úlohy.
4. Pokud cílovým není, pak ho expanduj a všechny jeho bezprostřední následníky vlož do fronty OPEN, pokračuj zpět na bod 2

Z tohoto slovního popisu lze snadno poznat, odkud má algoritmus svůj název. Strom dané úlohy slepě prochází, přičemž vždy projde celou úroveň stromu, než se odváží do další. To z něj činí velmi pomalý a paměťově náročný algoritmus, který se téměř nepoužívá.

Z praktického hlediska by ovšem nastal problém, protože v tomto případě, kdy uvažujeme pouze frontu OPEN, by docházelo k znovuexpandování již expandovaných uzlů. Tomu se dá jednoduše předejít přidáním seznamu, který se většinou nazývá CLOSED a obsahuje již expandované uzly. Prohledávání do šířky by se tedy změnilo následovně:

1. Sestroj frontu OPEN a seznam CLOSED. Do OPEN umísti počáteční uzel
2. Vyber z fronty OPEN první uzel a umísti ho do seznamu CLOSED, pokud ovšem žádný prvek ve frontě není, hledání řešení je neúspěšné, konec
3. Je-li vybraný uzel cílový, pak ukonči algoritmus s tím, že právě vybraný uzel je cílovým stavem dané úlohy.
4. Pokud cílovým není, pak ho expanduj a uzly, které nejsou ani ve frontě OPEN či seznamu CLOSED umísti do fronty OPEN a pokračuj bodem 2

2.1.2 Prohledávání do hloubky

Tento algoritmus má jistou podobnost s předchozím algoritmem, bohužel má ale jisté nepříjemné vlastnosti. Za prvé ho nelze považovat za úplný, což znamená, že není jisté, zda nalezne řešení a za druhé není optimální, což nezaručuje, že nalezené řešení je nejlepší možné. Metoda svého názvu docílila tím, že prohledává stavový strom do hloubky a ne po úrovních, jak tomu bylo v předchozím případě. Toho je dosaženo tím, že OPEN již není fronta, ale zásobník. Celý algoritmus pak vypadá následovně (seznam CLOSED se v tomto případě téměř nepoužívá, protože tím algoritmus ztratí savou lineární náročnost):

1. Vytvoř zásobník OPEN. Do OPEN vlož počáteční uzel
2. Vyber ze zásobníku první prvek, pokud je OPEN prázdný, pak neexistuje řešení, algoritmus skončil neúspěchem
3. je-li vybraný uzel cílový, pak algoritmus končí úspěchem
4. pokud není, pak ho expanduj a do OPEN umísti všechny jeho bezprostřední následovníky, kteří již v OPEN nejsou a nejsou předky expandovaného uzlu, pokračuj bodem 2

Jak bylo zmíněno výše, seznam CLOSED se téměř v tomto případě nepoužívá, protože jeho použitím se náročnost algoritmu změní z lineární na exponenciální. Jednou z možností, jak eliminovat znovugenerování stejných uzlů (kromě metody uvedené v algoritmu výše), je omezit prohledávání do určité hloubky. V tomto případě musíme mít alespoň nějaký předpoklad, jak hluboko může řešení maximálně ležet. Díky tomu pak můžeme zamezit dalšímu prohledávání do hloubky.

2.1.3 Metoda stejných cen

Poslední metodou, kterou bych rád zmínil před tím, než přejdu k metodám informovaným, je metoda stejných cest. Datová struktura OPEN je tentokrát reprezentována jako seznam. Hlavním rozdílem oproti podobné metodě prohledávání do šířky je, že metoda uvažuje skutečnou cenu cest. To je výhodné zejména pokud například vyhledáváme cestu z místa A do místa B a víme, kolik kilometrů jsme urazili přes jednotlivé cesty. Algoritmus metody stejných cen za použití seznamu CLOSED může tedy vypadat následovně:

1. Vytvoř dva seznamy OPEN a CLOSED. Do OPEN vlož počáteční uzel.
2. Vyber ze seznamu prvek, který má nejlepší ohodnocení a přidej ho do seznamu CLOSED, pokud v OPEN seznamu již žádný prvek není, pak algoritmus skončil neúspěchem
3. Je-li vybraný uzel cílový, pak algoritmus končí úspěchem
4. Pokud není, pak jej expanduju a všechny jeho následníky, kteří nejsou v seznamu CLOSED, přesuň do OPEN včetně jejich ohodnocení. Pokud je v OPEN jeden uzel vícekrát, pak v něm nech ten, který má nejlepší ohodnocení a ostatní vyškrtni. Pokračuj bodem 2

Výhody této metody by se měly projevit zejména v úlohách pro vyhledávání cesty. Proto by tento algoritmus připadal v úvahu pro náš konkrétní problém. Nejdřív se ale podíváme, co nabízí informované metody, které vše můžou zvrátit ve svůj prospěch.

2.2 Informované metody

Informované metody již svým názvem napovídají, že nabízejí oproti předchozím metodám něco navíc. Je to právě informace o cílovém stavu, a zejména pak mají prostředky, jak aktuální stav hodnotit. Pro naši potřebu jsou vhodné zejména algoritmy z rodiny metod založených na výběru nejlépe ohodnoceného stavu. Ty jsou velmi podobné metodě stejných cen, také používají cenu cesty, která je ale nyní určena součtem ceny přechodů a odhadované ceny cesty ze současného uzlu do uzlu cílového. Odhad je nazývan heuristikou. Zmíním se zde o dvou variantách této metody a to lačné vyhledávání (jedná se jen o volný překlad, dále se bude používat jen anglický originál Greedy Search) a A*.

2.2.1 Greedy Search

Metoda Greedy Search nepoužívá součet ceny cest, ale pouze heuristiku. Nebudu zde znovu popisovat algoritmus, který je téměř stejný jako u metody stejných cest. Ze seznamu OPEN akorát vybírá metodu s nejmenší heuristikou. Je očividné, že dobrá heuristika je pro tuto metodu vším a pokud je zvolena špatně, pak nemusí být metoda úplná, optimalni nebo bude mít neadekvátní časové nároky. Časové nároky lze nicméně značně zredukovat právě vhodnou volbou heurisitky.

2.2.2 A*

A* je metoda, kteará používá jak cenu cesty, tak odhad cesty z aktuálního uzlu do uzlu cílového. Heuristická funkce musí být tzv. spodním odhadem, což zaručuje úplnost, optimálnost a vynikající časové a prostorové nároky.

2.3 Výběr algoritmu

V této kapitole jsme si uvedli pět metod, zmínili jsme jejich výhody a nevýhody a teď nás čeká poměrně snadný úkol. Tím je určit, kterou metodu zvolíme pro naše potřeby, kterými je vyhledání cesty. Výběr padl právě na poslední metodu, která oproti ostatním téměř nemá nevýhody. V další kapitole ji detailněji popíšeme a zmíníme se o jejím využití v praxi.

3 Metoda A*

Metoda A* je informovanou metodou založenou na výběru nejlépe ohodnoceného stavu. Hlavním problémem této metody je, jak stav nejlépe ohodnotit. K ohodnocení se většinou používá vztah

$$f(n) = g(n) + h(n)$$

f(n) je celkovým ohodnocením uzlu n. g(n) je součet cen cest, které jsme během cesty do uzlu n urazili. A konečně h(n) je odhad ceny cesty z uzlu n do cílového uzlu. U metody A* tento odhad musí být spodním odhadem skutečné ceny cesty. Cena cesty v počátku g(0) je logicky nulová, heuristická funkce je nicméně nulová zase v cíl h(G).

3.1 Proč jsme vybrali metodu A*

A proč jsme právě metodu A* upřednostnili přede všemi ostatními metodami, když jsme vybírali metodu, která nám nejlépe pomůže vyřešit úlohu na vyhledávání cesty? Z neinformovaných metod připadal v úvahu snad jen metoda stejných cest, která je víceméně podobná algoritmu A*, její heuristika je ovšem nulová, což by v našem případě bylo vzhledem k tomu, že jsme heuristickou funkci schopni sestavit, poměrně neefektivní. Greedy search je zase metoda, která nevyužije to, že jsme schopni spočítat, kolik nás stojí cesta, což by vedlo ke zbytečným časovým a prostorovým nárokům. Jednoznačně tedy pro naši úlohu zvolíme právě A*.

3.2 Pseudokód algoritmu A*

Pseudokód algoritmu, který tedy použijeme, je následující:

1. Vytvoř seznamy OPEN a CLOSED
2. Přidej startovní uzel do OPEN seznamu
3. V OPEN nalezni buňku s nejmenším ohodnocením f
4. Přesuň ji do listu CLOSED
5. Pokud je tato buňka cílová, pak máme výsledek a rekurzivně jsme schopni najít cestu od startu do cíle
6. Jinak generuj všechny uzly, které mohou být dosaženy z uzlu aktuálního
7. Spočítej součet ceny cen do právě generované buňky (g)

8. Testuj, zda se aktuální stav nachází v seznamu OPEN. Pokud ano a jeho ohodnocení g je stejné nebo lepší jak uzlu aktuálně generovaného, pak pokračuj bodem 10
9. Testuj, zda se aktuální stav nachází v seznamu CLOSED. Pokud ano a jeho ohodnocení g je stejné nebo lepší jak uzlu aktuálně generovaného, pak pokračuj bodem 10
10. Smaž všechny výskyty stejného uzlu v seznamech OPEN a CLOSED
11. Vlož tento nově generovaný uzel do seznamu OPEN včetně jeho ohodnocení f
12. Vrať se na bod 3, dokud je v OPEN nějaká buňka

Jak lze vidět, opravdu je algoritmus metody A* velmi podobný algortimu metody stejných cest. Prakticky se skutečně jen liší v tom, jak je vypočteno ohodnocení.

3.3 Použití A* pro vyhledávání cesty

Výběr algoritmu A* pro naši aplikaci není zrovna náhodný a není ani nijak velkým překvapením. Pro úlohy, kdy je nutno vyhledat cestu, jsou jeho různé varianty velmi používané. V dnešní době populární GPS navigace využívá s výhodou tento algoritmus. Naprostá většina her, kde je nutno implementovat pohyb postav po herní mapě tento algoritmus používá také. Dokonce roboti, kteří touto dobou čím dal tím častěji objevují taje nám vzdálených vesmírných těles, používají modifikace této metody. Jak je vidět, výběrem metody jsme rozhodně neprohloupili.

4 Návrh aplikace

Nyní se konečně dostáváme k návrhu aplikace a přesněji specifikujeme problém, před kterým námi vybraná metoda A* stojí. Již jsem se několikrát zmínil, že se bude jednat o vyhledávání cesty, což je ovšem specifikace nedostatečná a neobjasňuje nám, co vše bude možno simulovat.

4.1 Specifikace mapy

Našemu algoritmu bude známa mapa, která se bude skládat z políček, jenž budou reprezentovány hexagony. Tím docílíme toho, že každé políčko, pokud nebude ležet na samém okraji mapy, bude mít přesně šest sousedů. Do nového stavu úlohy nás tedy algoritmus dostane přesunem na některého ze sousedů aktuálně expandovaného políčka. Políčka by mělo být možno adresovat v nějakém souřadném systému. To ovšem není problém, protože políčka jsou uspořádáná v matici. Základní rozdíl oproti klasické představě čtverce, který si představíme jako prvek matice, je v tom, že čtverec má čtyři nebo osm sousedů (podle toho zda uvažujeme i šikmý pohyb), kdežto hexagon má pevně daný počet šesti sousedů.

4.2 Váha políček

Aby vše nebylo tak jednoduché, jak se na první pohled zdá, tak dalším požadavkem na aplikaci je, že by měla obsahovat různé druhy terénu. Simuluje se tím tak cena jendotlivých přechodů, což pomůže algoritmu, aby se vyhnul nevýhodným typům terénu, jako může být například nerovná kamenitá cesta. Druhů terénu by mělo být několik. Od toho nejlépe průchodného, tím bude travnatá louka, přes obtížněji schůdné terény, jako jsou kamenité, až skalnaté cesty, po vyloženě neschůdné terény jakými jsou zasněžené pláně a hory. Algorimus musí tyto aspekty brát v úvahu a snažit se najít nejvýhodnější cestu, i když by se mohla zdát na první pohled delší než je třeba vzdušná vzdálenost.

4.3 Neprůchodné překážky

Dalším prvkem na mapě, na který může algoritmus A* při svém pilném prohledávání narazit, jsou neprůchodné překážky. Políčko, na kterém je takováto překážka umístěna, se stane nedosažitelné a algoritmus jej nemusí nikdy expandovat. Abychom pokračovali v realistickém zobrazení, tak tyto překážky budeme modelovat jako zeď, strom nebo husté křoví.

4.4 Krokování algoritmu

Jelikož má aplikace sloužit zejména jako didaktická pomůcka, pak je nutno, aby aplikace nebyla schopna jen vyhledat cestu, ale také názorně demonstrovat průběh algoritmu. Toho by mělo být dosaženo tím, že uživateli umožníme A* algoritmus provádět krok po kroku a dokonce se v něm i vracet. Pokud by se nám povedlo graficky odlišit buňky, které již byly expandovány a které teprve na svou expanzi čekají v seznamu OPEN, docílili bychom tím velmi dobré názornosti a použitelnosti.

4.5 Expanzní strom

Algoritmus A* při průchodu stavovým prostorem vytváří tzv. expanzní strom, který znázorňuje dosavadní průběh celého algoritmu. Jeho kořenovým prvkem je startovní políčko, kde začala naše strastiplná pouť po mapě. Kdyby se nám povedlo nějakým způsobem tento strom znázornit i v naší aplikaci, byl by to znovu velký krok kupředu, protože by tím šlo velmi detailně a přehledně sledovat, jak se algoritmus chová. Představa je taková, že model stromu a mapa by měli koexistovat vedle sebe a při kliknutí na kterýkoliv uzel ve stromu zvýraznit ekvivalentní políčko na mapě. Problém ovšem bude, jak tento strom znázornit, protože může být velmi objemný a v průběhu řešení algoritmu se může ztrácet výhoda přehlednosti a použitelnosti tohoto stromu. Je tedy naším cílem se na tuto konkrétní část výrazněji zaměřit a implementavat ji co nejefektivněji při zachování maximální možné míry ergonomie.

4.6 Editační a simulační část

Jak plyne z předchozích požadavků, tak by se aplikace měla skládat ze dvou hlavních částí. První část bude umožňovat editaci terénu, druhá pak simulaci podle zadaných kritérií v části editační. Tyto dvě části je nutno striktně oddělit, aby nebylo možno v průběhu simulace měnit terén a náš algoritmus tak zmást. Při bližším pohledu na editační část by vzhledem k předcházejícím podkapitolám mělo být jasné, co by vše měla zvládat. V první řadě by měla umožnit vložení počátečního a cílového bodu na mapě. Tyto body je nutno graficky zvýraznit tak, aby byly snadno viditelné. Další jeho schopností by měla být možnost měnit povrch terénu, tedy měnit travnaté louky na skály a zasněžené ledovce. Tím se určí váha jednotlivých přechodů. V neposlední řadě by měla editace umožnit vkládání neprůchodných překážek, které by měly být taktéž řádně graficky zpracované a dostatečně názorné. Nesmíme také zapomenout, že bychom rádi demonstrovali vliv heuristiky na efektivitu řešení. Proto bychom měli v této části uživateli umožnit, aby si mohl zvolit jednu z předem definovaných heuristik.

V simulační části konečně pustíme algoritmus metody A* ke slovu a budeme sledovat, jak se popere ze zadaným terénem. V nástrojové liště simulace by rozhodně neměly chybět prostředky,

které budou umožňovat krokování algoritmu, což bude zřejmě vyžadovat dvě tlačítka, která umožní jak provádět kroky algoritmu, tak se v něm vracet zpět. Také bychom měli umožnit, aby jedním kliknutím uživatel rovnou vyhledal celou cestu a nemusel se tak zdžovat postupným krokováním. Bude to užitečné zejména v případě, kdy uživatele zajímá skutečně jen cesta terénem, který si nachystal.

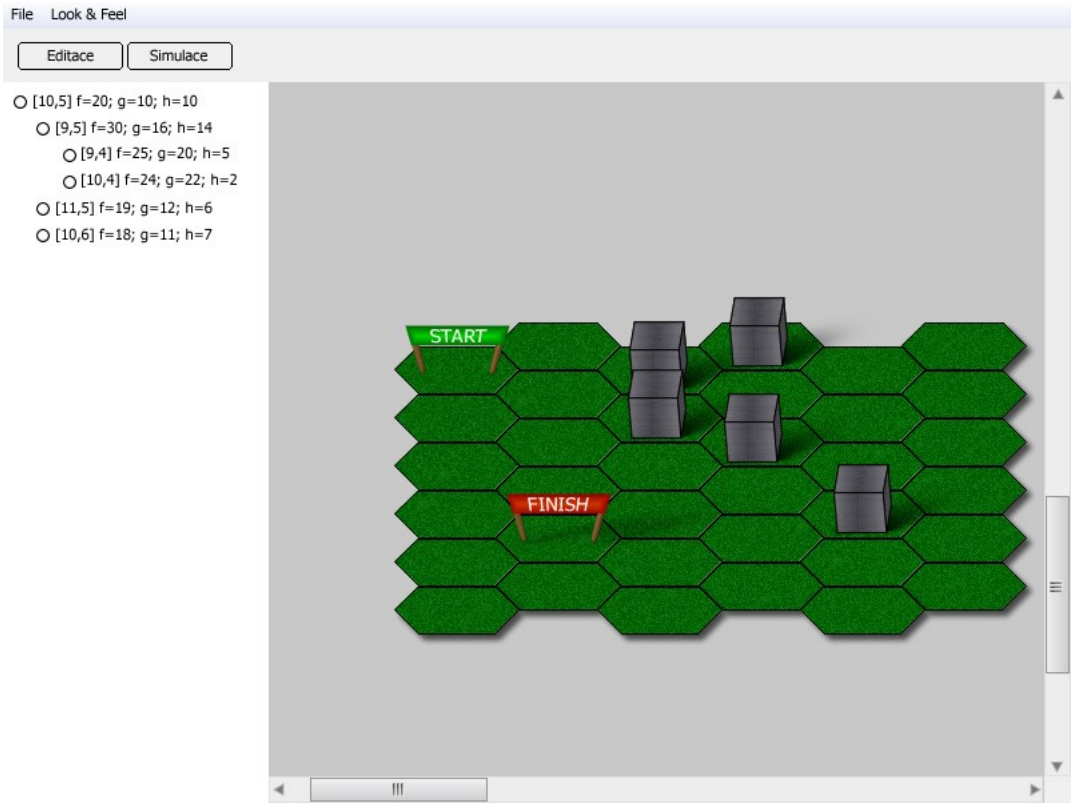
Jedním z hlavních požadavků bylo, aby uživatel mohl přehledně sledovat, jak se postupem času vyvíjí expanzní strom. Proto se při každém provedení kroku, ať už dopředu nebo nazpět, musí tento strom aktualizovat. To zaručí skutečné pochopení toho, jak algoritmus funguje.

4.7 Heuristiky

Aplikace má za úkol mimo jiné demonstrovat, jak velký má vliv heuristika na průběh algoritmu. Původně jsem zamýšlel zahrnout několikero heuristik, většina z nich by se ovšem lišila od těch ostatních minimálně. Proto jsem nakonec usoudil, že heuristiky budou stačit jen dvě. Musí se ovšem lišit dost významně. První z nich, která je pro úlohy na vyhledávání cesty typická, je vzdušná vzdálenost od daného políčka do cíle. Mapa je uspořádaná do matice, takže spočítat vzdálenost lze triviálně pythagorovou větou. Jako druhou heuristiku jsme zvolil nejhorší možnost heuristické funkce a tou je nulová heuristika. Ta degraduje metodu A* na odnož metody stejných cen. Nalezené řešení bude tedy také optimální, jeho náročnost ale bude ve většině případů daleko větší. To by mělo, vzhledem k možnosti algoritmus krokovat a prohlížet si expanzní strom, dostatečně demonstrovat důležitost heuristiky u metody A*.

4.8 Návrh grafického uživatelského rozhraní

Přehledné uživatelské rozhraní a atraktivní grafika je alfou i omegou toho, zda aplikace uživatele zaujme a on se k ní bude rád vracet. Nepřehledný a graficky ošklivý program mohl zaujmou před deseti lety, kdy bylo důležité, aby program hlavně fungoval, ovšem v dnešní době se nároky zvyšují a ve velké konkurenci jsou to právě tyto zdánlivé drobnosti, které odliší program od konkurence a doslova ho prodají.



Obrázek 4.1: Návrh GUI pro applet

Stál jsem před problémem, jak implementovat nemalou funkčnost a zachovat přesto relativní jednoduchost ovládání. Pokud by student musel věnovat u výukového programu ještě nemalé úsilí tomu, aby se program naučil vůbec používat, pak by asi celá aplikace brzy skončila na smetišti dějin. Proto jsem se při návrhu snažil, aby největší část obrazovky zabírala ta nejzajímavější část a tou je mapa terénu. Jak lze vidět na obrázku výše, zabírá drtivou část okna aplikace a jednoznačně vštípí do podvědomí uživatele, že tato část je ta nejdůležitější. Vedle mapy bych umístil část, která bude mít sice své opodstatnění jen při simulaci, ale přesto je pro aplikaci také nepostradatelná. Tato část bude právě zobrazovat expanzní strom, který se bude v průběhu simulace rozrůstat, proto je potřeba zajistit, aby uživatel mohl měnit pohled na tuto část, což by měl zajistit jezdec. Dále bychom měli

uživateli umožnit, aby mohl plynule měnit poměr prostoru, který zabírá strom vůči mapě. Editační a simulační nástroje bych pak umístil nahoru, kde by mezi nimi mohl uživatel přepínat.

Tímto se dá považovat návrh za kompletní a nic nám nebraní v tom, abychom se pustili do nejzajímavější části projektu a tou je samotná implementace.

5 Implementace projektu

Jelikož je můj bakalářský projekt zejména implementační, tak bych této části také rád věnoval největší porci prostoru.

5.1 Výběr programovacího jazyka a nástrojů

Když mi v hlavě vznikala myšlenka na tento projekt, měl jsme ve výběru programovacího jazyka poměrně jasno. Jazyk C++ mi připadal dostatečně flexibilní, aby se s podobným projektem vypořádal naprosto bez problémů. Navíc jsem se při studiu na Fakultě informačních technologií poměrně dobře seznámil s nástrojem wxWidgets, který jsem byl odhodlán také použít. Pro jazyk C++ ještě mluvila rychlost a hlavně možnost objektově orientovaného stylu programování, které jsem měl v plánu využít.

Tahle představa se mi velmi rychle rozplynula, když jsem poprvé navštívil svého vedoucího, který byl spokojen s výběrem projektu, nicméně jazyk C++ mi rozmlouval. Zejména se mu nelíbilo to, že aplikaci bych mu dodal jako spustitelný exe soubor a kdykoliv by jej chtěl použít, musel by jej mít po ruce. Dal jsem mu za pravdu a nakonec jsme se domluvili na jazyku Java s tím, že aplikaci vytvořím jako applet, který bude spustitelný přes webový prohlížeč. Z tohoto výběru jsem zpočátku nebyl zrovna dvakrát nadšen, nicméně jak jsem Javu lépe poznal, byl jsem svému vedoucímu vděčný za to, že mi původní záměr s jazykem C++ rozmluvil. Ocenil jsem zejména to, že používání objektů je v tomto jazyce velmi přirozené a jeho syntaxe se příliš od jazyka C++ neliší, což mi velmi usnadnilo jeho naučení.

Pro ovládací prvky jsem zvolil komponenty, které nabízejí standardní knihovny Swing. Původně jsem se chtěl jakémukoliv integrovanému prostředí vyhnout. Jak ale postupně projekt nabýval na své velikosti, připadalo mi psaní a správa projektu čím dál víc těžkopádnější a složitější. Proto jsem začal ke své velké spokojenosti volně dostupné vývojové prostředí Netbeans, které mi mnohé usnadnilo. Vynikající se mi zejména zdála nápověda, která mi při mých malých zkušenostech s Javou velmi pomohla. Nemalé potíže mi ušetřil i vizuální návrh prostředí, který navíc generoval velmi čistý kód.

Pro umístění appletu na web jsem chtěl původně použít jednoduché statické stránky psané v HTML. To se ale postupem času, kdy na applet byly kladeny další pořadavky, ukázalo jako nedostačující. Jelikož v dnešní době obřích monitorů bylo původní rozlišení 800 na 600 obrazových bodů absolutně nedostačující, musel jsem sáhnout po nástroji, který by byl schopen předat appletu parametry a měnil tak svou velikost. K tomuto účelu se mi jevil jako dostatečný makrogenerátotr PHP.

5.2 Mapa

Jak už bylo několikrát zmíněno, mapa je pro tento program tím nejpodstatnějším. Její dobré grafické ztvárnění přitáhne pozornost. Špatný vzhled naopak odsoudí aplikaci k neúspěchu. Mapa je potomkem swingové třídy JPanel, což se v konečném důsledku projevilo zejména v rychlosti, která není nejlepší, ale pořád dostatečná. Java obecně má známé problémy z rychlostí. K tomu když se přidá programátor, který není v Javě zrovna zběhlý, tak dostáváme rychlost aplikace, která rozhodně nemůže porazit aplikace psané v jiných jazycích.

Jak bylo zmíněno v návrhu, mapa je v paměti reprezentována jako matice, v níž má ovšem každý prvek maximálně jen šest sousedů. Přímo v aplikaci se pak projevilo, že políčka s lichou x-ovou osou musí být oproti těm sudým posunuta o polovinu své výšky. Tím dosáhneme toho, že se mapa bude jevit velmi konzistentně a značně realisticky. Bohužel tady jsem poprvé skutečně narazil. Aby mapa vypadala dostatečně realisticky a šlo na ní rozeznat jednotlivé detaily, musí být graficky vyvedena na příslušné úrovni. To jsem po pár neúspěšných pokusech o vlastní tvorbu vzdal a usoudil, že pokud má aplikace vypadat skutečně atraktivně, musím se uchýlit k cizí pomoci. Naštěstí jsem nemusel chodit daleko a kolega grafik, s kterým už několik let vyvíjíme amaterské počítačové hry, mi velmi ochotně přislíbil, že nějakou grafiku vytvoří. Jak je vidět z obrázků a samotné aplikace, povedlo se mu to opravdu dobře a tímto bych mu rád ještě jednou poděkoval.

A co z jeho povedené práce plyne? Kromě toho, že celkový dojem je o několik úrovní vyšší, než v případě aplikace s mojí grafikou, tak má onen líbivý vzhled i značný význam informativní. Na první pohled lze odhadnout, jaký je druh terénu i bez toho, aniž by si uživatel četl manuál.

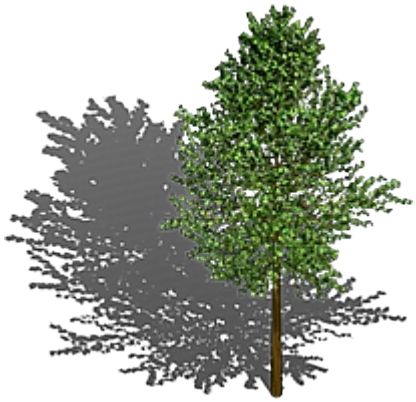
Tím se plynule dostávám k tomu, jaké druhy terénu jsou v aplikaci dostupné. Zvolil jsem nakonec pět druhů, které podrobněji popíše následující tabulka:

	Travnatá louka	Váha = 1
	Štěrková cesta	Váha = 3
	Skalnatý masiv	Váha = 5
	Zasněžená step	Váha = 7
	Zasněžené hory	Váha = 10

Tabulka 5.1: Přehled druhů terénu

Jak lze vidět, potvrdila se má slova, že grafika je velmi názorná. Jednotlivé váhy terénu značí, jak moc je náročný přechod přes tento terén. Například přechod přes jedno políčko, kde se nacházejí zasněžené hory je stejně náročný, jako přechod přes deset políček, kde se nachází travnatá louka.

Podobná situace byla i u objektů, které jsou ztvárněny velmi realisticky a myslím, že i velmi pěkně. Původně jsem chtěl, aby byl výběr jen z jednoho objektu, a to prosté cihlové zdi. Nakonec se ale ukázalo, že by se tím ztratily výhody, které poskytuje pět druhů terénu a mapa by tak vypadala příliš fádně. Následující tabulka ukáže, že jsem se nakonec rozhodl pro objekty tři:

	Prostá betonová zeď
	Listnatý strom
	Keř

Tabulka 5.2: Přehled objektů

Objekty opět vypadají velmi realisticky. Dokonce, pokud uživatel umístí strom či keř na zasněžený povrch, ztratí rostliny své listy. Objekty mají především bránit průchodu přes políčko a to jejich grafická reprezentace znázorňuje víc než dostatečně.

Nakonec bylo potřeba vytvořit objekty, které by značily, kde je start a kde je cíl cesty na mapě. Jejich grafická reprezentace je následující:

	Cedule, která znázorňuje start
	Cedule, která znázorňuje cíl

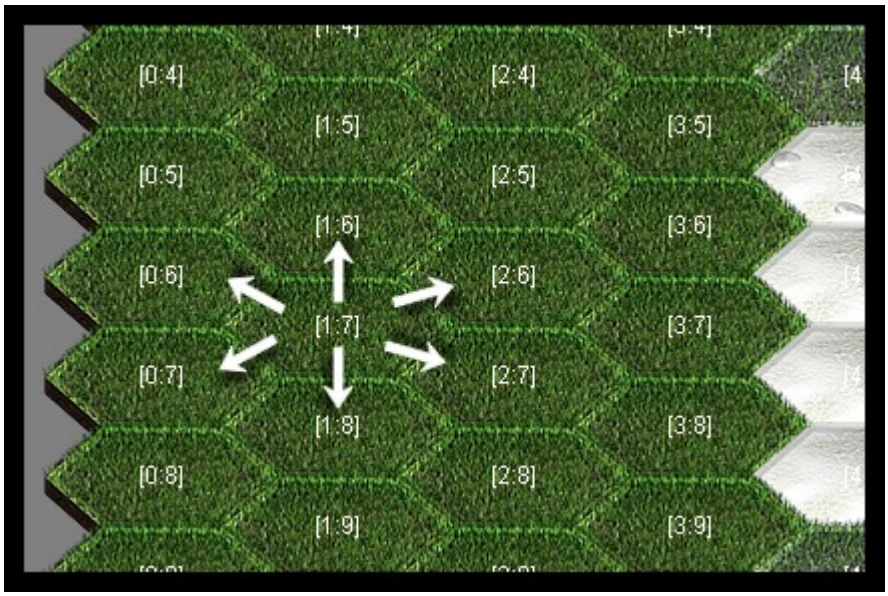
Tabulka 5.3: Objekty start a cíl

Pastelově výrazné barvy cedulí jsou dostatečně vyditelné i na velmi zastavěné mapě, kde je spousta objektů. Při implementaci také bylo nutno myslet na to, aby uživatel nemohl umístit objekty přes sebe. Algoritmus by nefungoval správně pokud by například uživatel umístil cílovou ceduli na políčko se stromem.

5.3 Reprezentace políček v paměti

Na tomto místě bych se rád zmínil o tom, jak jsem implementoval jednotlivá políčka mapy. Ta jsou totiž reprezentována jako objekty. Nechci se pouštět do detailního popisu všech proměnných a metod, které políčko obsahuje, ale zmínil bych alespoň ty nejdůležitější, které mají podstatný vliv na správný průběh při vyhledávání cesty.

Jak jsem již několikrát zmínil, celá mapa je složena s hexagonů, takže každé políčko může mít maximálně šest sousedů. Stál jsem ale před problémem, jak správně určit, které sousedy dané políčko má, aby algoritmus věděl, jak má políčko expandovat.



Obrázek 5.1: Možnosti pohybu

Jak lze vidět na obrázku, políčka mají vždy souseda spodního a horního. Boční sousedi se ovšem liší v závislosti na tom, zda se políčko nachází na liché či sudé x-ové souřadnici. Políčka o sudé x-ové souřadniuci (včetně nuly) mají levého a pravého horního souseda o stejné y-ové souřadnici, jako políčko samotné. Leví a praví spodní sousedé pak mají hodnotu této osy o jednu menší. Horní sousedé políček, která se nacházejí na liché x-ové souřadnici, jsou naopak o přesně jednu jednotku výš, než jsou políčka samotná. y-ová osa spodního levého a pravého souseda je pak ekvivalentní hodnotě výchozího políčka. Proto není jednoduché vždy jednoznačně určit sousedy

daného políčka. Každé políčko je v paměti reprezentováno jako instance třídy políčko. Aby expanze políčka zabrala vždy minimum času a vyžadovala minimum úsilí, tak jsem se rozhodl, že každé políčko bude mít šest ukazatelů na své okolní sousedy. Proto se navázání sousedů každému políčku provede jen jednou, při inicializaci mapy. Tím docílíme toho, že sousedé každého políčka budou vždy velmi pohotově po ruce a není nutno je zbytečně vyhledávat. Pokud je políčko na kraji a nemá tedy souseda, je příslušný ukazatel prázdný.

Každé políčko si také drží údaj o své váze, která je dána druhem terénu. Ten je možno měnit v editačním módu. Cena cesty přes toto políčko je tedy násobena tímto údajem. Podobná situace je také s ukládáním informace, která překážka se nachází na daném políčku. Algoritmus se pak při pokusu o expanzi políčka dotáže, zda je na políčku nějaká nepřekročitelná překážka a pokud je tato podmínka pravdivá, tak nedojde k umístění do seznamu OPEN.

Každé políčko má také zabudované prostředky, jak usnadnit algoritmu A* jeho práci. Jedná se zejména o proměnné, v kterých se uchovává hodnota ceny cesty G a hodnota heuristicke funkce H. Kdykoliv algoritmus potřebuje tyto údaje, nemusí je znovu počítat. Každé políčko, které bylo expandováno, má také o svém rodiči, kterým byl expandován. To usnadňuje jak výstavbu a hlavně grafickou reprezentaci expanzního stromu, ale také je tento údaj velmi užitečný při rekonstrukci finální cesty od startu do cíle. Poslední proměnná, který strojí za zmínku, je logická hodnota, která určuje, zda je políčko finální cestou. To se hodí, pokud algoritmus skončí úspěchem a my bychom rádi zobrazili výslednou cestu. Ta se určí tak, že z cílového políčka, které algoritmus A* nalezl, se vrátíme skrze expanzní strom zpět k políčku startovnímu. Políčka, která takto navštívíme, označíme, abychom mohli jednoduše vykreslit cestu. Jde vidět, že pomůcek, jak usnadnit práci algoritmu máme několik, z toho ovšem plyne, že nesmíme zapomenout tyto proměnné náležitě nulovat, pokud algoritmus začíná znovu. Mohlo by se totiž stát, že by výsledek předchozí simulace negativně ovlivnil simulaci novou.

5.4 Samotná mapa

Pokud bychom ale fungovali jen se samotnými políčky, brzy bychom asi narazili na značnou neohrabanost tohoto přístupu. Proto jsou všechna políčka „zastřešena“ dalším objektem, jehož třída má všeříkající název „MainMapa“. Její nejvýznamější částí je pole ukazatelů na jednotlivá políčka mapy, čímž tak logicky sjednocují políčka podobným způsobem jako mapa reálná nebo jako ta, kterou jsme měli vymyšlenou při návrhu. Dále obsahuje údaje o šířce a výšce mapy, což je užitečné kdykoliv, kdy je potřeba projít celou mapu. Na to je totiž potřeba údaje o mezích mapy, což zaručují právě tyto hodnoty.

Třída mapa také obsahuje dva ukazatele na políčka, která jsou startem respektive cílem řešeného problému. Je důležité mít tyto ukazatele vždy po ruce, protože v průběhu algoritmu k nim budeme několikrát přistupovat.

Po dlouhém váhání jsem se rozhodl, že metody, které jsou potřebné pro vyřešení úlohy vyhledání cesty, umístím právě do této třídy. Jestli je to rozhodnutí nejlepší si nejsem jistý, určitě je to ale přehlednější, jelikož v této třídě máme vše potřebné k tomu, abychom mohli konečně začít s implementací algoritmu samotného.

5.5 Implementace A* algoritmu

Po sáhodlouhých přípravách se konečně dostáváme k tomu, proč tento demonstrační program vzniknul. Jelikož máme vše potřebné velmi dobře připraveno, nebudeme mít snad s implementací vyhledávání cesty žádné problémy.

Jelikož jsem se výše zmínil, že metody algoritmu A* budu implementovat jako součást třídy *MainMapa*, musím do této třídy přidat ještě některé potřebné datové struktury, které algoritmus bude potřebovat ke své činnosti. Mluvím zejména o dvou seznamech s názvy OPEN a CLOSED, o kterých byla řeč už mnohokrát a jejich funkce by nám měla být zřejmá. Pro jejich reprezentaci užijeme instance standardní javovské třídy vector, která poskytuje velmi širokou paletu metod. Hodně z nich využijeme i my, protože vkládání uzlů, jejich odebírání a vyhledávání bude velmi frekventovanými operacemi.

Jelikož bylo jedním z požadavků algoritmus krokovat, základní metodou bude krok algoritmu A*. Ten je zároveň srdcem celého algoritmu. Hlavní úlohou této metody je nalézt uzel s nejmenším ohodnocením v seznamu OPEN, přesunout ho do seznamu CLOSED, otestovat ho, zda není uzlem cílovým a pokud ano, pak zastaví jakoukoliv další činnost a zobrazí cestu. Pokud ne, tak ještě expanduje aktuálně vybraný uzel. Pro každý z nových uzlů spočítá cenu dosavadní cesty. Poté otestuje, zda se daný uzel nenachází již v seznamu OPEN. Pokud ano, zjistí, zda uzel v OPEN nemá horší ohodnocení, jak uzel expandovaný a pokud tomu tak je, tak uzel z OPEN odebere a tváří se, jako by jej v seznamu OPEN ani nenašel. To samé se opakuje se seznamem CLOSED. Se seznamem CLOSED je to složitější, protože pokud je zvolená heuristika, která je spodním odhadem, pak by situace, kdy v seznamu CLOSED je uzel s horším ohodnocením, neměla nastat. Nesmíme ovšem zapomenout, že máme možnost zvolit nulovou heuristiku, která této podmínce neodpovídá a mohl by tak nastat výše zmíněný případ. Pokud ale uzel není ani v OPEN ani v CLOSED, pak spočítáme pomocí heuristické funkce odhad. Uzel pak spolu s takto vypočteným odhadem a cenou cesty uložíme do seznamu OPEN.

Metoda, která implementuje krok algoritmu A* je stěžejní metodou pro dvě hlavní funkcionality programu a tím je jednorázové nalezení cesty a krokování tohoto problému. Pro první případ je krok metody A* opakován do té doby, než je nalezena cesta k cíli nebo krok metody vrátí chybu, protože seznam OPEN je prázdný. V druhém případě (režim krokování) se metoda kroku zavolá jen jednou a to je vše. Neplatí to ovšem pro první krok, kdy je potřeba inicializovat oba seznamy a nastavit ostatní parametry. Rozdílné jsou také stavy, ve kterých se bude zobrazovat mapa po ukončení z těchto dvou metod. Při kompletním průchodu se na ní zobrazí jen výsledná cesta. Pokud ovšem krokujeme, pak se na mapě barevně zvýrazní políčka respektive uzly, které se právě nacházejí v expanzním stromu. Ta z nich, která se nacházejí v seznamu OPEN budou zvýrazněna modře. Políčka v seznamu CLOSED budou pro změnu vybarvena červěně. To zaručuje maximální přehlednost a kontrolu nad tím, co algoritmus právě provádí. Po nalezení cesty se ovšem opět zvýrazní jen cesta (žlutým obarvením). Není ovšem problém se v algoritmu opět vrátit a dostat se tak do stavu těsně před nalezením cíle.

Krokování zpět byl vůbec u takového lineárního algoritmu celkem oříšek. Spoléhal jsem ovšem na to, že algoritmus není příliš náročný na výpočet. Proto si vždy s každým novým krokem algoritmu uchovávám celkový počet kroků. Pokud se uživatel přeje vrátit o krok zpět, pak provedu o jeden méně kroků algoritmu, než mám uloženo. Tím se zaručuje požadovaná funkcionalita při zachování dostatečné rychlosti.

Ostatní metody jsou víceméně pomocné a pokud bych se o nich dále zmiňoval, přesahovalo by o rámec této dokumentace.

5.6 Vykeslování mapy

Programová logika je již hotová a nezbývá nic jiného, než ji atraktivně a přehlědně ztvárnit vizuálně. Původně jsem si myslel, že tahle část bude pomyslnou „třešničkou na dortu“ celého projektu. Bohužel Java mi z něj udělala doslova peklo. Přitom způsob provedení nebyl odlišný od toho, jaký je prezentován na většině výukových stránkách jazyka Java.

Kreslit jsem se rozhodl na potomka swingové třídy JPanel, který jsem příznačně pojmenoval *DrawPanel*. Tato třída by měla obsahovat ukazatel na instanci třídy mapy, pomocí nějž bude poté mapu vykreslovat. Snažil jsem se ctít veškerá doporučení a proto se jednodlitvé obrázky nevykreslují přímo na plátno kreslícího panelu, ale kreslím si je nejdříve do pomocého obrázku, který teprve, až je hotov, vykreslím. Tato metoda se nazývá double-buffering a měla by zamezit blikání, které by mohlo vznikat. Také by tato metoda měla být rychlejší, protože se tím minimalizuje režie, která nastává při jakémkoliv kreslení na swingovou komponentu. Můžu potvrdit, že double-buffering znatelně

vykreslování urychlil, oproti původnímu řešení, kdy jsem každý obrázek vykresloval přímo. Bohužel pořád to zřejmě nestačilo.

Jak tedy samotné vykreslování mapy probíhá? Jako první se musí vykreslit povrch mapy a teprve poté se na ni vykreslí všechny obekty. Z toho tedy plyne, že vykreslování bude dvouprůchodové. To je nutné právě kvůli tomu, že vše, co vykreslujeme, jsou předrenderované obrázky s alfa průhledností. Objekty obsahují mimo jiné i stín, který navozuje větší pocit realičnosti. Kdybychom však vykreslovali vždy políčko i s objektem, tak by se mohlo stát, že nám další políčko tento stín překryje.

Další problém, na který jsem narazil, byl v tom, že některá políčka jsou travnatá. Původně, když jsem políčka vykresloval směrem zleva doprava po řádcích shora dolů, tak se stávalo, že políčka na lichých pozicích překrývala trávu svého levého souseda kvůli svému relativnímu posunu vůči políčkům sousedním.To vyústilo ve velmi nepěkný efekt. Proto je potřeba vykreslit nejprve políčka na lichých pozicích a teprve poté všechna sudá políčka. Tím se dosáhne toho, že terén vypadá pěkně souvisle. Obrázky políček jsou dostupné vždy ve dvou verzích a to s ohraničením či bez. Uživatel si může zvolit, který z těchto obrázků se má vykreslit a pokud si vybere ten s ohraničením, tak se tím simuluje mřížka mapy. Při průchodu cykly, kterými se povrch mapy vykresluje, je vhodné rovnu vykreslit i jeho případné obarvení. K tomu dochází v případě, že se dané políčko nachází v seznamech CLOSED nebo OPEN, popřípadě je součástí cesty ze startu do cíle.

Pokud je již vykreslen terén, pak nesmíme zapomenout umístit na něj druhým průchodem i všechny nezbytné objekty. Výsledný efekt je zejména právě díky stínům velmi realistický.

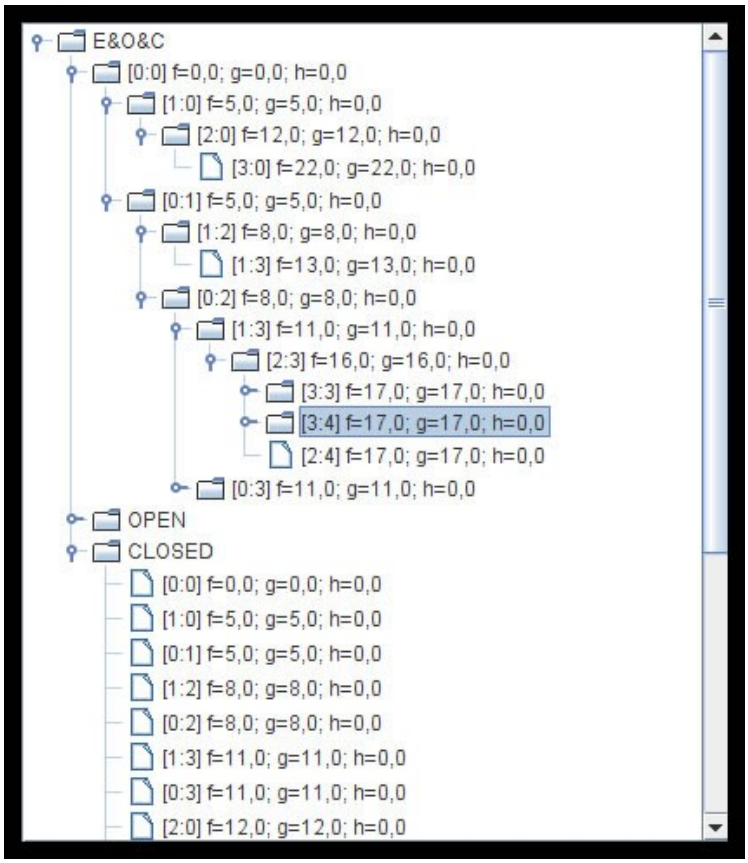
Java umožňuje měnit velikost obrázků před jejich samotným vykreslením. Toho jsem s výhodou využil a implementoval tak možnost zmenšovat a zvětšovat měřítko mapy, což může být velmi užitečné.

Veškeré ovládání, které souvisí s mapou je implementováno přímo v této popisované třídě. Jsou to zejména metody pro posun mapy, přibližování a oddalování a editace terénu podle aktuálně zvoleného druhu vkládání (viz další kapitola)

5.7 Expanzní strom

Nedílnou součástí celé aplikace je právě expanzní strom, který je implementován jako potomek swingovské třídy JPanel a k samotnému zobrazení expanzního stromu používá všech výhod standardní třídy JTree, která umožňuje zobrazovat data v stromové hierarchii s ohledem na jejich skutečnou hierarchii fyzickou.

Celým stromem pak jde velmi přehledně procházet s ohledem i na velmi rozsáhlé případy. Je rozdělen na tři hlavní části. Tou nejdůležitější je samotný strom, kde každý prvek tohoto stromu je instancí objektu další námi zavedené třídy. Tato třída obsahuje ve své definici údaje o pozici, heuristice a ceně celkové cesty konkrétního uzlu. Tyto informace slouží ke dvěma účelům. Zejména



Obrázek 5.2: Expanzní strom

se z nich skládá popis uzlu v tomto stromu, což slouží k rychlé a snadné orientaci. Je to dále prostředek k tomu, abychom mohli určit skutečné políčko na mapě, na které uživatel v expanzním stromu klepnul myší. To se na mapě projeví tak, že nad oným políčkem se zobrazí šipka. Tento mechanismus slouží ke snadnějšímu určení, které políčko na mapě je ekvivalentní s právě vybraným uzlem.

Strom mimo samotný expanzní strom také zobrazuje stav seznamů OPEN a CLOSED. U těchto dvou seznamů je také možnost zobrazit na mapě ekvivalentní políčko.

5.8 Dokončení práce

Tímto máme téměř vše hotovo a nám zbývá jen vše poskládat dohromady. Vzhledem k tomu, že program vyvíjíme v prostředí NetBeans, je tvorba uživatelského prostředí velmi jednoduchá. Do jednotlivých palet standardních komponent lze přidávat i komponenty vlastní. Budeme se řídit podle návrhu, který jsme si vytvořili minulou kapitolu. Dva hlavní prvky, expanzní strom a zobrazení mapy umístíme do komponenty JSplitPane, která nám umožní, že lze měnit poměr mezi prostory, které oba prvky zabírají. Nad tento panel umístíme další JPanel, ve kterém se budou nacházet tlačítka a combo-boxy pro editování terénu a simulaci. Přepínání mezi těmito dvěma módy umožní záložky. Poslední částí naší aplikace bude spodní JPanel, který bude obsahovat tlačítka pro rychlé rozbalení a zabalení stromu a zaškrtávací pole pro volbu, zda chce uživatel zobrazit mřížku nebo pozice polí na mapě. Posledním úkolem, který nám zbývá, je navázat jednotlivé funkcionality prvků na ploše na události jako jsou stisky tlačítek apod. Java, potažmo NetBeans nám k tomu však poskytuje velmi komfortní nástroje, takže je to to nejmenší. Ve zkratce to lze popsat tak, že instancím jednotlivých tříd přiřadíme prvky, které se jmenují *listenery*.

Posledním slavnostním krokem je vytvoření balíčku s příponou JAR, který je vhodný k umístění na internet, protože je již komprimovaný a neklade tak velké nároky na rychlost připojení uživatele.

5.9 Umístění na internet

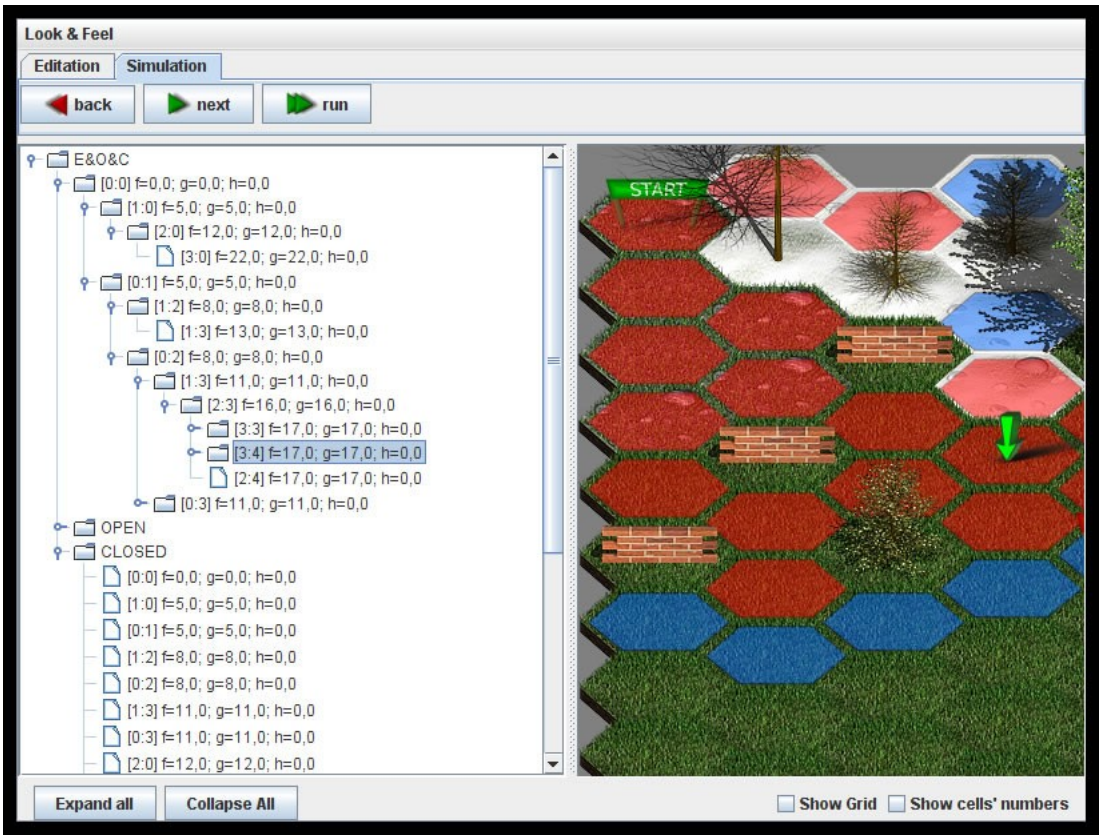
Nyní nás čeká umístění balíčku na internet. PHP skript je velmi jednoduchý. Generuje pouze HTML kód, který předá našemu appletu parametry, jenž slouží k nastavení velikosti vykreslovací plochy pro mapu. Skript jsem pak jen umístil spolu s JAR balíčkem a obrázky na svůj hosting.

5.10 Testování

Aplikaci jsem ještě také podrobil testování, které mělo prokázat, zda je aplikace skutečně spustitelná na všech počítačích, jak bylo původně v plánu. Bohužel se ukázalo, že tento požadavek se nepodaří splnit, protože aby bylo applet skutečně možno spustit, je nutno mít nainstalované běhové prostředí Javy ve verzi 1.6.x. Na nižších verzích se mi aplikaci nepovedlo spustit. Jinak jsem se přesvědčil, že ve správně nakonfigurovaných prostředích běží aplikace bez problémů.

Na obrázku *5.3* lze již vidět výslednou aplikaci v akci. V editačním režimu jsem upravil terén a následně se přepnul do simulačního rozhraní. Zde jsem spustil vyhledání cílové cesty a následně se

vrátil o krok zpět. Tím jsem dostal úlohu do stavu, který bezprostředně předchází konečnému řešení úlohy.



5.3 Aplikace v simulačním režimu

Také lze pozorovat, jak úzce souvisí část obsahující expanzní strom s částí, kde je znázorněna mapa. V tomto případě se to, že uživatel vybral uzel v stromu, projevilo na mapě tím, že se nad ekvivalentím políčkem objevila šipka. V seznamech OPEN respektive CLOSED lze také prohlížet všechny uzly určené k expanzi respektive uzly uzavřené.

6 Závěr

Cílem této práce bylo vytvořit výukovou aplikaci, která by měla pokud možno získat studentovu pozornost. To je v dnešní době, kdy má student za sebou už několik hodin přednášek, velmi obtížné a stál jsem před nelehkým úkolem. Myslím ale, že se mi to podařilo, zejména díky stylu, v jakém je aplikace graficky provedena. Hlavní část, zobrazení mapy, totiž není nepodobná některým hrám a studentovu roztěkanou pozornost hned zaujme. To dává přednášejícímu možnost demonstrovat tuto velmi zajímavou problematiku plně soustředěnému přednáškovému sálu. Netvrdím pochopitelně, že podobná aplikace by se na celém internetu nenašla, ale co jsem měl tu možnost, tak jsem nenalezl aplikaci, která by byla líbivější při zachování stejných didaktických schopností. Přínos aplikace tak doufám nebude jen lokální, ale pokud se mi práci povede obhájit, rád bych se o ni podělil se světem, protože si myslím, že bude mít úspěch a pomůže mnoha programátorům pochopit problematiku vyhledávání cesty. V projektu ale nevídím jen přínos pro ostatní, ale i osobní. Zejména jsem vděčný svému vedoucímu, který mi doporučil jazyk Java a já tak využil příležitost tvorby tohoto projektu, abych se s tímto velmi populárním jazykem seznámil. Musím říct, že jeho styl mi velmi vyhovoval. Také jsem rád, že jsem sám danou problematiku prohledávání stavového prostoru detailně pochopil a uviděl v ní obrovskou perspektivu a využití v praxi.

Projekt splnil všechny požadavky, které si dal za cíl. To ovšem nepopírá pravidlo, které říká, že žádný projekt nikdy není úplně hotový a vždy obsahuje nějaké chyby. Aplikaci bych později rád rozšířil o možnost výběru metody prohledávání stavového prostoru. Metoda A* se sice jeví jako jasně nejlepší, to ovšem neznamená, že by nešlo využít i jiných mechanismů. Dále bych také rád zapracoval na rychlosti. Přiznávám, že vykreslování mapy je na slabších počítačích už nepříjemně pomalé a bylo by pošetilé svalovat vinu jen na Javu, u které je sice známo, že její běhové prostředí není zrovna nejrychlejší. V neposlední řadě bych také časem rád doladil zdrojový kód aplikace, který doplácí na mou nezkušenost s tímto vyspělým a komplexním jazykem. Zejména bych se víc zaměřil, aby kód více ctil zásady objektově orientovaného progamování.

Literatura

[1] Russel, P., Norvig, S. , *Artificial Intelligence - A Modern Approach, Prentice Hall, 2002*

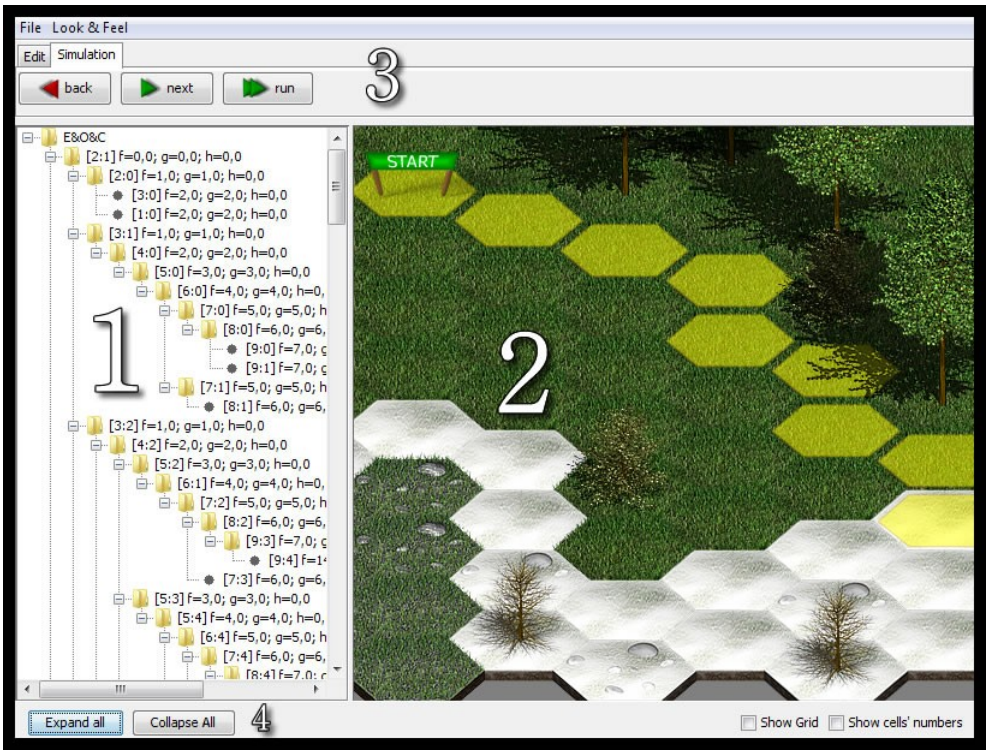
[2] Zbořil, František., *Opora do předmětu IZU* [online].
Dostupný z WWW: <https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-4.pdf>

[3] Sun Microsystems, *The Java Tutorials*
Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/>

Seznam příloh

Příloha 1 – Manuál k ovládání aplikace

Ovládání aplikace je velmi jednoduché a intuitivní, přesto ho velmi stručně popíšu.



Aplikace je rozdělena do čtyř základních částí.

1. **Expanzní strom** Zde se zobrazuje aktuální expanzní strom. Klepnutím na uzel se na mapě zobrazí ekvivalentní uzel
2. **Mapa** Nejdůležitější část aplikace ovládána převážně myší
   Posun mapy – za neustále stisknutého levého tlačítka se hýbe myší
   Změna měřítka – kolečko myši nahoru a dolů
   Editace terénu – nejprve si vybereme typ vkládání v části 3 a poté vkládáme stiskem levého tlačítka myši
3. **Editace/simulace** Mezi jednotlivými módy se přepínáme klepnutím na záložku
   Editace – cathegory je typ objektu, který se projeví v nabídce item, kde dojde k výběru konkrétního objektu, který bude vkládán. Nabídka heuristika slouží ke zvolení příslušné heuristiky

Simulace - Tlačítka back a next slouží ke krokování algoritmu. Tlačítko Run jednorázově najde cestu

4. **Další nastavení** Tlačítka, která se nachází vlevo, tj. Expand All a Collapse All slouží k rychlému rozbalení respektive zabalení stromu. Show Grid zobrazí nebo skryje mřížku mezi políčky na mapě. Show cells' number slouží k zobrazení souřadnic políček v části mapy.