Implementace A* algoritmu na konkrétní problém orientace v prostoru budov

Popis problému

Orientaci ve známém prostředí lze převést na problém nalezení cesty z místa A do místa B. Obecně platí, že robot vykonávající takovou cestu se na místo určení může dostat několika různými způsoby. V reálných příkladech nás však zajímají řešení splňující následující cíle:

- Jak se dostat z bodu A do B
- Jak se vyhnout překážkám na cestě
- Jak najít nejkratší možnou cestu
- Jak najít příslušnou cestu rychle

Existují algoritmy splňující některý nebo i všechny tyto body, nebo nesplňují žádný
Problém se značně komplikuje, když robot pohybující se v takovém prostředí má nebo nemá „božské“ znalosti o světě, ve kterém se pohybuje. Z tohoto hlediska můžeme algoritmy na hledání cest rozdělit na dvě velké skupiny:

Informované hledání cesty

Při tomto hledání cesty má robot kompletní informace o prostoru, ve kterém se pohybuje a zná přesné místo, kam se má dostat. Zásadní nevýhodou je nutnost co nejrozsáhlejšího průzkumu terénu ještě před vlastním výpočtem cesty. Hledání cesty při znalosti prostředí se používá velice často, nejmasovější rozšíření těchto algoritmu se používá v počítačových hrách.

Využití:

- V počítačových hrách
- Pomoc postiženým lidem při orientaci v budově
- Hledání cest na mapách
- Vojenská technika

Neinformované hledání cesty

Robot na začátku nemá žádné informace o svém okolí a musí se zcela spoléhat na vlastní zdroje informací a průzkum terénu. Nezná tedy celou cestu, ale hledá ji na základě informací získaných ze svého okolí. Tyto poznatky může shromažďovat a učit se na základě již získaných zkušeností.
Cena za nedostatek informací je ovšem ta, že se ve většině případů musíme spokojit s prostým nalezením cesty.

Využití:

- Například automatické sondy na Marsu. Kvůli zpoždění při komunikaci se sondou, musela být vybavena programy řešící některé situace při pohybu bez pomoci základny.
- Záchranářská technika
- Vojenská technika

Hledání cest v počítačových hrách

U algoritmů, které se používají v počítačových hrách, se k požadavku nalezení cesty připojují i další kritéria, jako:

- **Rychlost a nenáročnost výpočtu.** Je potřeba zajistit, aby počítač byl schopen propočítat cestu pro velké množství robotů, kteří se mohou v některých hrách vyskytovat.
- **Inteligence pohybu.** Je potřeba najít takovou cestu, aby se roboti ve hře pohybovali „inteligentně“ tak, aby hráč měl pocit, že roboti nějakou inteligenci mají
- **Interakce mezi hráčem a roboty nebo mezi roboty navzájem**. Při hledání cesty se musí zohledňovat aktuální stav hry.

Poslední dva požadavky značně zvyšují náročnost algoritmu, jsou ovšem v rozporu s prvním požadavkem. Algoritmy používané ve hrách jsou proto hlavně kompromisem mezi nimi.

Svět vytvořený v počítačové hře má na rozdíl od skutečného světa jednu základní výhodu. Můžeme si ho totiž sami vytvořit tak, aby co nejvíce vyhovoval danému algoritmu. Je konečný. Můžeme si před spuštěním programu některé věci přepočítat nebo přednastavit tak, aby samotný výpočet byl co nejrychlejší.

Využití neinteligentních algoritmů

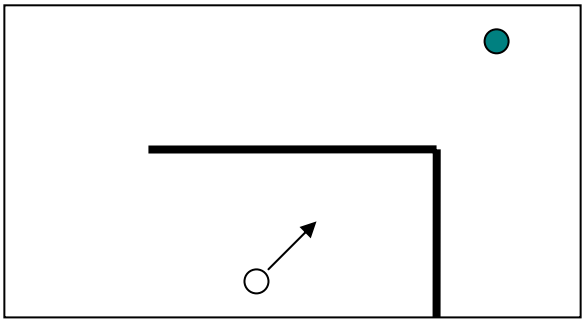
Někdy je vhodné z hlediska výkonu využít ten nejjednodušší algoritmus. Jejich další výhodou je, že robot nemusí znát detailně své okolí, stačí mu jen základní informace, typu dál nemůžu, přede mnou je zeď. Tyto algoritmy se také dají použit pro orientaci v neznámém prostředí.

Algoritmus „Přímo k cíli“

Nejjednodušší algoritmus vypočítá směr k cíly a po této přímce se robot vydá. Tento algoritmus nefunguje na členitém terénu. Dojde-li robot ke stěně zarazí se a nemůže pokračovat dál.

Tento algoritmus je ve většině případů nevhodný. Obzvlášť pokud se cíl nehýbe, v takovém případě se robot k cíly, pokud narazí na překážku, prostě nedostane. Pokud se ale cíl (hráč) pohybují v bludišti, může takto „zaseknutý“ robot působit, jako že na hráče „číhá“ někde za rohem.

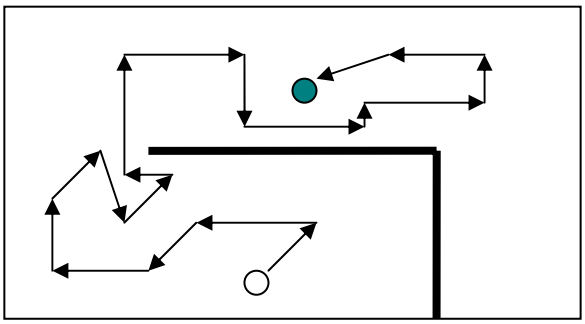
V každém případě je tento algoritmus základní, protože ostatní algoritmy mohou rozdělit cestu na množinu bodu, mezi kterými nejsou žádné překážky a robot pak vykonává cestu mezi těmito body.



Robot směřuje přímo k cíly,ale pokud hráč neobejde stěnu, nikdy se k němu nedostane

Algoritmus „Náhodného pohybu“

Poněkud zvláštní algoritmus založený na myšlence, když nevím kudy, prostě někudy vyrazím a někdy se k cíli prostě dostanu. Tento algoritmu se dá použít i v případě, že robot nezná prostor, ve kterém se pohybuje. Navíc lze rozšířit o řadu heuristik, které zvýší pravděpodobnost a rychlost nalezení cesty.



Robot se náhodně pohybuje, dokud nenajde hráče

Využití algoritmů se znalostí prostředí

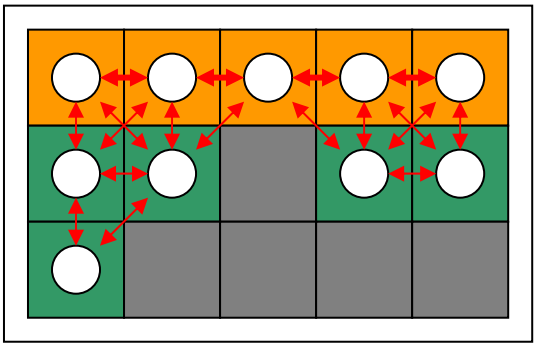
Známe-li prostředí, ve kterém se robot pohybuje, můžeme jej převést na graf, kde uzly jsou křižovatky a hrany cesty, po kterých se může robot pohybovat. Uzly a hrany ohodnotíme funkcemi, které vypočítají jejich cenu vzhledem k různým okolnostem.
Existuje celá řada možností, jak tento algoritmus dále rozšířit.
Například vytvořit několik grafů a jejich ohodnocení odpovídají různým situacím, které mohou ve hře nastat a pro různé skupiny robotů v závislosti na požadavcích, které na ně máme, můžeme přidělit různé grafy. Můžeme tak například zařídit, že roboti symbolizující ve hře „útočné komando“ se budou pohybovat v postranních uličkách a na otevřené prostranství půjdou, až když nebude jiná možnost.
Další důležitý aspekt je, že si můžeme potřebné grafy vygenerovat do zásoby, takže při hře nezatěžují tolik procesor. Navíc se mohou výsledky z rozpočítané cesty použít pro více robotů pohybujících se v prostředí hry.

Generování grafů

Při tvorbě prostředí v počítačových hrách jde logiku tvorby prostředí rozdělit do dvou základních skupin. Podle toho, kterou reprezentaci světa programátor zvolí, se řídí princip tvorby grafů.

1) Dlaždicové světy

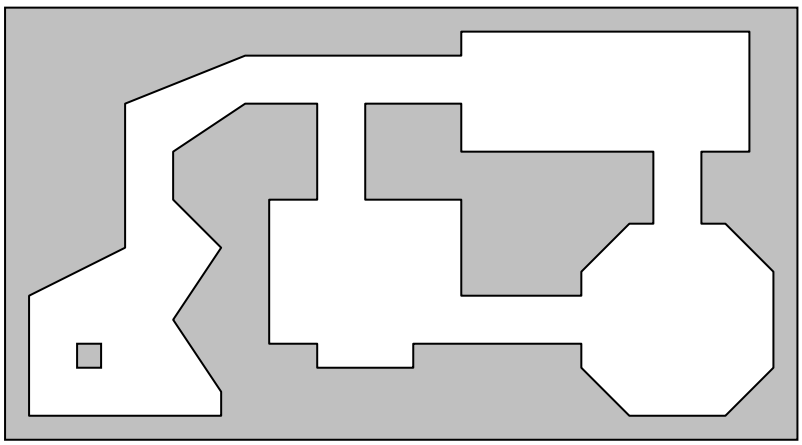
Tyto světy jsou dobře známé ze starších, převážně strategických her, kde byl herní plán rozdělen na čtverce nebo šestiúhelníky.
Jednotlivá pole tvoří uzly grafu a hrany mezi dlaždicemi. Mohou být ohodnoceny čísly představující náročnost přechodu z jednoho pole na druhy, dají se tak velmi pěkně vytvořit třeba cesty, po kterých se budou roboti pohybovat častěji a rychleji než třeba v lese.
Další výhodou takto vytvořeného světa je jeho snadná dynamická změna. Pokud nějaké dlaždice změní svoji hodnotu, je velmi snadné přetvořit graf tak, aby tuto změnu zachytil.



Jednoduchá mapka s cestou lesem a skalou.

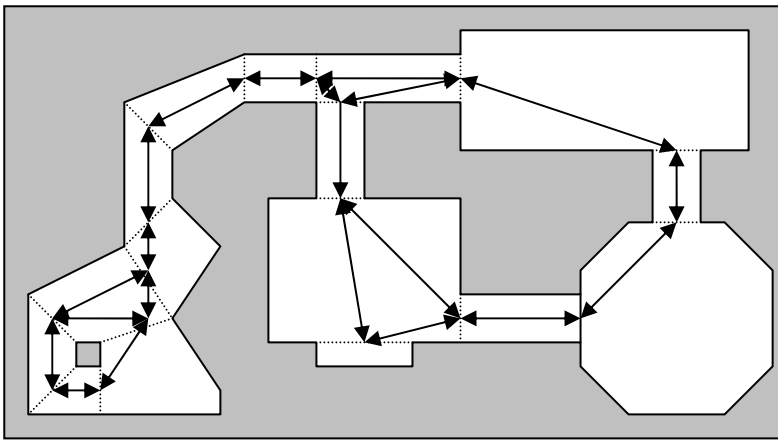
2) Světy tvořené vektory

Ve většině moderních her, jsou ale prostředí, ve kterých se roboti pohybují tvořeny vektorovými objekty. Musíme tedy do plánu takto tvořeného světa rozmístit uzly a hrany tak, aby graf zachycoval podstatu plánu a přitom, aby žádná hrana neprocházela stěnou.Pak bychom totiž nemohli použít jednoduchý algoritmus pro přesunutí robota z místa A do B. To se dá udělat ručně nebo automaticky. To znamená,



Plánek budovy tvořený vektory

že musíme plán rozdělit na konvexní polygony. Jejich hrany pak rozdělit na stěny a portály. Tím zaručíme, že robot pohybující od portálu k portálu nikdy nenarazí do stěny. Je zde ještě jedna malá podmínka a sice, že portály, mezi kterými se robot pohybuje, nesmějí ležet na jedné přímce. V takovém případě by robot narazil do stěny mezi nimi. Dalším problémem je, že startovní a cílové místo prohledávání se obvykle nenachází v portálu, ale díky konvexnosti místností (polygonu rozdělujících plán) se robot dokáže přesunout od portálu ke každému místu v místnosti a naopak. Pro algoritmus, který pak generuje graf to znamená, že musí před každým hledáním cesty do grafu přidat počáteční a koncový uzel.

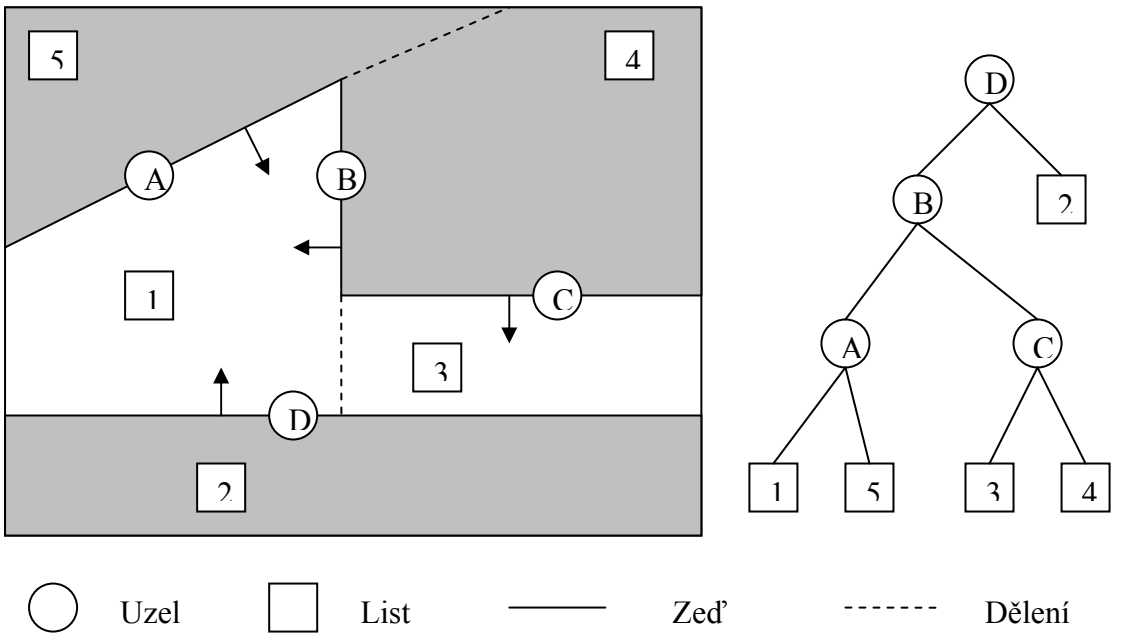


Plánek budovy rozdělený s portály a hranami grafu

Jak tedy tvořit plány vyhovující těmto kritériím a pokud možno automaticky v nich rozmisťovat portály?
Jedním z řešení je už průběhu tvoření plánu použít BSP stromy.

Co to jsou BSP stromy a jak nám pomůžou při generování portálů

BSP stromy (binary space partitioned tree) je technika správy polygonů v prostoru tak, aby umožnila jejich snadné vykreslování. Umožňuje také snadno detekovat kolize a automaticky tvořit portály. BSP také splňuje podmínky binárního vyhledávacího stromu.
Každý uzel představuje rozdělení prostor na dně části a každý list tohoto stromu pak představuje jednu oblast. Takto lze prostor rozdělit na konvexní polygony a snadno v něm vyhledávat.



Příklad mapky, jejího rozdělení a BSP stromu

Máme-li prostor takto uspořádaný pomocí BSP stromu, pak nám stačí projít listy stromu a přidat do nich portály. Hrany grafu jsou pak spojnice mezi všemi portály patřící jedné místnosti. K ohodnocení hran pak můžeme požít funkci vzdálenosti mezi portály.

3) Jiná schémata

Může existovat celá řada schémat světů a principů, jak v nich generovat světy a kombinace mezi nimi.

Použití prohledávacího algoritmu

Konečně se dostáváme k použití A* algoritmu. Ten potřebuje ke svému fungování odhadovací funkci. Nicméně jedna, celkem kvalitní, je přímý odhad vzdálenosti vzdušnou čarou od místa výskytu robota k jeho cíly. Vzhledem k povaze plánu, který neobsahuje takové rušivé elementy, jako jsou patra, je tato hodnotící funkce vyhovující.

Výhody a nevýhody použití A* algoritmu při prohledávání cesty.

Samotný princip hledání cesty v prostoru má řadu dalších problémů, se kterými se musí programátor vypořádat. Zde uvádím některá z nich

1. **Tvar vypočítané trajektorie a chování robotu.** Roboti, kteří se pohybují po takto vypočítaných drahách, se občas chovají v rámci požadavku na dojem jisté umělé inteligence poněkud zvláštně.
    a. Tvar trajektorie se skládá z přímek, to působí podivně, protože roboti zatáčejí pouze v místech portálů. To si vynucuje ještě dodatečné přepočítání dráhy a její aproximaci pomocí křivek => zvýšení nároku na určení cesty.
    b. Roboti se mohu rozhodnout vydat k nejbližšímu portálu místo, aby zamířili přímo k cíly
2. **Změna prostředí** nebo pohyb cíle si vynutí přepočítání celé cesty. V praxi se tento problém dá vyřešit tak, že roboti neustále počítají svoji cestu dokola. To zase může nepříjemně zatěžovat procesor. Nehledě na to, že vytvoření nového grafu se může zase stát nezadatelnou režií.
3. **Graf se v průběhu výpočtu mění** A* si do grafu ukládá informace o uzlech, které již prošel, to znamená, že k jednomu grafu nemohou zaráz přistupovat dva algoritmy. Při větším množství robotů a rozsáhlých grafech nepříjemně roste jak paměťová tak výpočtová náročnost.
4. **Všichni roboti pohybující se podle jednoho grafu mají stejnou trajektorii.** Je zřejmé, že všichni roboti hledající cestu podle jednoho grafu, najdou vždy stejnou cestu.

Přikládám konkrétní implementaci v javě
AStarNode.java, AStarSearch.java

Orientace v počítačovém modelu budov

Základní rozdíl mezi počítačovým modelem reálné budovy a labyrintem počítačové hry je v tom , že plán budovy nelze tak jednoduše rozložit na konvexní polygony . Rozhodneme-li se tedy použít algoritmus založený na hledání cesty v grafu, musíme vymyslet speciální převod plánu budovy do grafu.

Příklad užití algoritmu pro orientaci v budově

Jeden z pěkných příkladů, kdy se hodí použít algoritmy založené na A* prohledávání grafu, je pomoci zrakově postiženým lidem při orientaci ve složitých prostorách, jako je například tato budova. Na naší fakultě v laboratořích HCI běží jeden projekt, který si klade právě tento cíl.

Při hledání optimální cesty v takových příkladech musí nalezená cesta splňovat mnohem náročnější požadavky.

1. **Její skutečná délka,** je sice důležitý faktor, ale není nejdůležitější. Mohou nastat situace, kdy je lepší upřednostnit delší cestu vedoucí po chodbě bez překážek, než provést dotyčnou osobu skladem plným překážek.
   Řešení: z těchto důvodu je potřeba jednotlivé místnosti rozdělit do kategorií, podle toho, jak je bezpečné, nebo i společensky vhodné danou místností procházet. Algoritmus by se pak měl snažit dané místnosti obcházet. Toho se dá docílit tak, že se k množině hran, které reprezentuji danou místnost přiřadí vyšší ohodnocení než mají okolní místnosti dohromady.
2. **Počet prošlých místností.** Z lidského subjektivního hlediska je lepší cesta, který projde méně místností. Toho lze docílit kombinací A* algoritmu s prohledáváním do šířky nebo přidáním podmínky, která při průchodu do další místnosti připočítá k hodnotě cesty penalizaci.
3. **Specifické požadavky uživatele.** Uživatel může mít jiné specifické požadavky. Někdy se uživatel může rozhodnou, že mezi patry v budově se bude pohybovat jen pomocí výtahu.
   Řešení: zde vstupuje na scénu algoritmus pro tvoření grafu, který musí umět z grafu budovy některé hrany a uzly dočasně odstranit a nebo, což je vhodnější jim vnutit nějakou hodnotu. V našem případě pokud uživatel nechce použít schody, se hodnota hrany, která reprezentuje schody nastaví na nekonečno (hodnotu vyšší než je součet hodnot všech ostatních hran) . Tak se zaručí, že uživatel pojede výtahem, ale pokud výtah není, bude muset po schodech.
4. **Závislost na právech uživatele.** Uživatel naopak někdy nemůže projít některou částí budovy, protože tam nemá fyzicky přístup. Pak ovšem už musí být příslušné hrany a uzly z grafu dočasně odstraněny.
5. **Propojení s budovou**. Bylo by pěkné, kdyby systém realizující tento úkol, mohl čerpat aktuální informace o budově a podle nich upravovat vyhodnocování cesty..

Ohodnocovací a odhadovací funkce

Ohodnocovací funkce musí zohledňovat výše uvedené požadavky. Navíc můžou existovat další zohledňující například členitost zdí ….
Naproti tomu odhadovací funkce může zůstat. Drobné vylepšení získáme, když k přímé vzdálenosti přičteme konstantu za každé patro mezi odhadovaným uzlem a cílem.

Reprezentace výsledků

Finální fáze převedení získané cesty do podoby, ve které je nejvíce stravitelná pro člověka. U nás se to řeší pomocí zařízení, které umožňuje hmatem si „osahat“ objekt v počítači.

Budeme-li procházet modelem budovy pouze v počítači, máme problém jednodušší, než kdybychom ho chtěli použít v robotovi, který se v reálné budově musí opravdu pohybovat. Ten se totiž musí umět vypořádat s neočekávanými situacemi, jako je třeba zatarasená cesta.

Zdroje:

- V první řade vlastní zkušenosti při implementovaní algoritmu pro procházení budovou, podřeny řadou dokumenty z internetu, jejichž adresy jsem bohužel už nenašel.
- http://www.brackeen.com
- http://wikipedia.org
- http://www.scienceworld.cz