České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Plánování trajektorií ve virtuálním 2D prostředí

Zdeněk Kasl

Vedoucí práce: Ing. Milan Rollo, Ph.D.

Studijní program: Elektrotechnika a informatika strukturovaný bakalářský

Obor: Kybernetika a měření

srpen 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Zdeněk K a s l

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný

Obor: Kybernetika a měření

Název tématu: Plánování trajektorií ve virtuálním 2D prostředí

Pokyny pro vypracování:

1. Seznamte se s algoritmy pro plánování trajektorií pro pohyb osob a jejich využitím ve virtuálních prostředích (multiagentní simulace, počítačové hry).
2. Vzájemně porovnejte vlastnosti algoritmů, popište pro jaké typy úloh a prostředí jsou vhodné, porovnejte je z hlediska výpočetní a paměťové náročnosti.
3. Popište omezení jednotlivých algoritmů a jejich vliv na efektivitu algoritmů, jako např. dynamika prostředí, kinematika objektů, diskretizace prostoru, přechod k 2.5D prostředí, využití pro simulace pohybu velkého množství objektů, apod.
4. Ve zvoleném programovacím jazyku naimplementujte vybrané algoritmy a experimentálně ověřte jejich vlastnosti.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Milan Rollo, Ph.D.

Platnost zadání: do konce zimního semestru 2011/2012

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S. -4-

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 29. 11. 2010

Poděkování

Rád bych tímto poděkoval Ing. Milanu Rollovi, Ph.D., vedoucímu bakalářské práce, za cenné rady, vstřícnost a ochotu.

Abstrakt

Bakalářská práce je zaměřena na problematiku plánování trajektorií ve virtuálním 2D prostředí, se zaměřením na plánování pohybu osob v multiagentních simulacích a počítačových hrách. V první části práce jsou teoreticky popsány přístupy používané při plánování pohybu objektů, omezení plánovacích algoritmů a jejich aplikace. Druhá část práce je zaměřena na praktické ověření vlastností běžně používaných metod diskretizace a vybraných plánovacích algoritmů. Ve zhodnocení provedených experimentů jsou uvedena doporučení pro volbu vhodné metody plánování pohybu.

Abstract

The bachelor thesis deals with trajectory planning in virtual 2D environment with main interest in motion planning for human–based objects in multi–agent simulations and computer games. The first part of the thesis theoretically describes common approaches to motion planning, applications and limitations of the algorithms. Second part of the paper is focused on experimental verification of theoretical behaviour of selected approaches to motion planning. The experiment results can be used for correct selection of motion planning approach.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, SW, projekty atd.) uvedené v přiloženém seznamu.

V Praze dne 15.8.2011 ..... Kasl Z. .....

podpis

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Plánování pohybu je velice rozšířený problém, který je třeba řešit v mnoha, často dosti odlišných aplikacích, konkrétně v mobilní robotice, dopravě, počítačových hrách a dalších. Vzhledem k různorodosti uplatnění lze k pohybu trajektorií přistupovat dvěma způsoby, dle povahy konkrétní aplikace lze objekt navigovat ve spojitém prostředí, nebo plánovat cestu v diskrétním stavovém prostoru. Ve virtuálních aplikacích jsou převážně používány plánovací algoritmy nad diskrétním stavovým prostorem. Cílem plánovacího algoritmu je nalezení optimální trajektorie z počátečního do cílového bodu dle daných kritérií (např. cena, spotřeba, vzdálenost, bezpečnost atd.) tak, aby byl objekt schopen tuto trajektorii absolvovat. Zejména jedná-li se o objekty se složitější kinematikou nebo dynamikou, je nutné při plánování uvažovat i tyto vlastnosti objektu. V tomto případě je také nutné rozlišovat mezi plánováním cesty a trajektorie.

Kromě metod prohledávání a diskretizace stavového prostoru hrají při plánování značnou roli i vlastnosti prostředí, jako je jeho proměnlivost, dimenze stavového prostoru vzniklého diskretizací, nebo výskyt 2,5D křížení. Zmíněná 2,5D křížení představují například mosty, nebo tunely, kde je objekt vázán k určité ploše, ale objekty se mohou pohybovat ve více rovinách nad sebou. Zatímco změny v prostředí komplikují úlohu prohledávání, 2,5D prostředí je náročnější po stránce diskretizace.

Tato práce se zabývá algoritmy pro plánování pohybu osob ve virtuálním 2D prostředí, tedy v multiagentních simulacích, nebo počítačových hrách. Jedním z možných uplatnění, kde plánování pohybu hraje důležitou roli je modelování virtuálních, inteligentně se chovajících bytostí. Cílem je vytvořit programovou kopii člověka, která jej bude svým vystupováním co možná nejvěrněji připomínat. Konkrétní aplikací pak mohou být například obyvatelé virtuálního města, nebo protihráči v počítačových hrách.

První část této práce představuje obecný úvod do teorie plánování pohybu. Kapitola 2 je rozdělena na plánování cesty a plánování trajektorie. Jsou zde popsány běžně používané metody pro plánování cesty ve spojitém prostředí, metody diskretizace prostředí a algoritmy pro plánování cesty v diskrétním prostředí. Problematika plánování trajektorie se týká spíše objektů se složitější kinematikou, než je tomu u virtuální osoby, a proto je zde pouze nastíněna.

V druhé části, tedy v kapitolách 3 a 4 jsou popsány především praktické aplikace a omezení plánovacích algoritmů. Je zde detailněji popsáno jak lze provádět plánování pohybu virtuálních osob, nebo jaké problémy do problematiky plánování vnáší proměnlivé prostředí.

Poslední část, kapitola 5, je zaměřena na praktické ověření vlastností vybraných plánovacích algoritmů a jejich vzájemné porovnání.

2 Úvod do teorie plánovacích algoritmů

K řešení problematiky plánování pohybu lze přistupovat dvěma způsoby, v závislosti na typu řešené úlohy je možné plánovat jak ve spojitém, tak v diskretizovaném prostředí. Spojité prostředí známe z běžného života, je definováno v každém bodě. Hranice mezi dvěma stavy, pokud využijeme přiblížení z oblasti plánování pohybu, jsou omezeny lidským vnímáním, nikoli prostředím jako takovým. Spojité prostředí tedy lze definovat jako nekonečnou množinu všech stavů[1] a to i na konečně velkém prostoru. Oproti tomu diskrétní prostředí je definováno jako konečná množina všech stavů a přechodů mezi stavy. Nutnou podmínkou ale je, že plocha kterou se snažíme popsat je konečně velká. Diskretizací spojitého prostředí jsou určeny hranice mezi stavy, tím ovšem dojde ke ztrátě a zkreslení části informací které obsahuje spojité prostředí. Diskrétní prostředí je hojně využíváno, protože umožňuje použití prohledávacích algoritmů.

Dále je možné při plánování pohybu rozlišit plánování cesty a plánování trajektorie. Při plánování cesty nejsou uvažovány kinematické ani dynamické vlastnosti objektu a nalezená cesta je množina stavů, které vedou z počátečního do cílového stavu, nezávislá na čase. Cestu je možné upravit v trajektorii použitím dalších algoritmů, nebo ošetřit pohyb podle nalezené cesty na nižší úrovni. Oproti tomu při plánování trajektorií jsou uvažována i kinematická a dynamická omezení objektů. Nalezená trajektorie přesně popisuje pohyb objektu v čase. Trajektorie je nutné plánovat tam, kde je vyžadována vyšší přesnost pohybu, nebo v aplikacích, kde je nutné zaručit realizovatelnost naplánovaného pohybu.

2.1 Plánování cest ve spojitém prostředí

Jedna z metod plánování pohybu ve spojitém prostředí, která je využívaná i ve virtuálních aplikacích, je metoda plánování cesty pomocí potenciálových polí. Jak již název napovídá, princip metody spočívá ve vytvoření virtuálního potenciálového pole, jehož vektor směřuje od počátečního do cílového bodu. Překážky jsou reprezentovány potenciálovým polem, jehož vektor směřuje směrem od překážky. S výhodou je možné definovat další pole, například takové, které bude měnit svou intenzitu podle změn v terénu. To způsobí, že se objekt bude pohybovat po nejsnazší cestě. Další pohyb objektu v každém bodě prostoru je dán váženým součtem všech potenciálů působících v tomto bodě.

Situaci si lze představit jako pohyb elektronu v elektrickém poli, kde je počáteční bod reprezentován silným záporným nábojem a cíl je naopak nabit kladně. Tím je vytvořeno hlavní potenciálové pole, které zajišťuje pohyb objektu směrem k cíli. Překážky si pak lze představit jako záporně nabitá místa, která deformují výsledné pole a zakřivují dráhu

[1] Stavem je v případě plánování pohybu myšlena poloha objektu v souřadnicovém systému.

pohybu tak, aby nedošlo ke kolizi.

Při použití v nekomplikovaném prostředí je nalezení cesty pomocí této metody rychlé a efektivní. Slabou stránkou tohoto přístupu k plánování pohybu je možnost uváznutí v lokálním minimu. Problém je možné vyřešit zavedením pomocného pole, které začne působit v případě, že je detekováno uváznutí objektu. Pomocné pole je uvažováno do doby, než je objekt mimo dosah lokálního minima.

Podrobnější informace o metodě navigace pomocí potenciálových polí lze nalézt v práci [22], kde se H. Safaldi zabývá především implementací metody pro účely navigace mobilního robotu.

2.2 Plánování cest v diskrétním prostředí

2.2.1 Diskretizace

V této podkapitole jsou popsány nejpoužívanější metody pro automatické vytvoření stavového prostoru pro účely plánování cesty. Pro vypracování byl použit článek D. Hale a kolektiv [7], článek F. Heckel a kolektiv [8], webové stránky [20] a online článek [23].

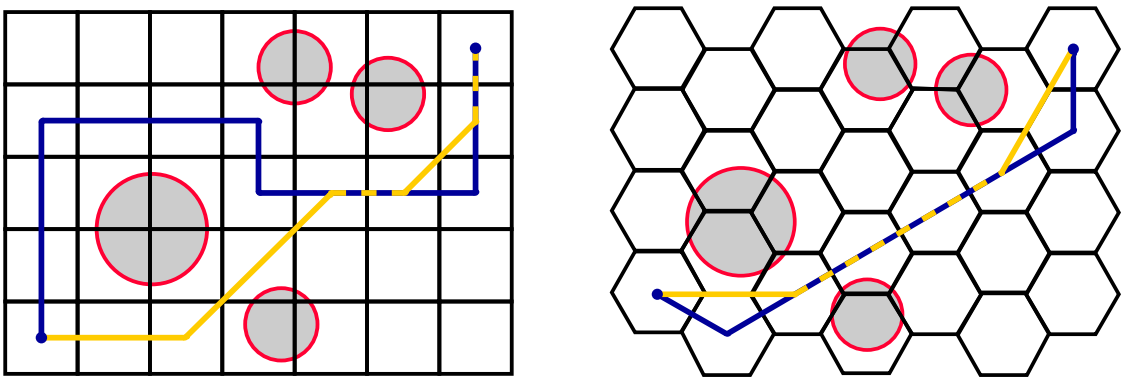
Prostředí je možné také diskretizovat ručně. Uživatel při běhu, nebo ještě před načtením prostředí do aplikace zvolí důležité stavy a vzájemně je propojí. Výsledný *way–point* graf popisuje dané prostředí. Výhodou této metody je, že uživatel může zvolit pouze důležité stavy.

2.2.1.1 Navigační mřížky

Tato metoda diskretizace je jednou z nejužívanějších. Principem metody diskretizace pomocí navigačních mřížek je, že se celý prostor pokryje mřížkou složenou z buněk definovaného tvaru. Nejpoužívanější jsou čtvercové, nebo šestihranné buňky. Lze použít i trojúhelníkové buňky, ale operace s nimi jsou náročnější, proto se pro účely plánování pohybu příliš nepoužívají.

Jednotlivé buňky mřížky představují diskrétní stavy, je tedy možné určit povahu konkrétní buňky (například povrch) a cenu, s jakou se lze přes buňku pohybovat. Pokud je buňka označena za neprůchodnou, jsou vynechány přechody vedoucí do příslušného stavu, který ji reprezentuje, nebo tento stav není vůbec vytvořen.

Čtvercové mřížky, viz obrázek 2.1(a), jsou používány především pro jednoduchost, s jakou s nimi lze zacházet. Čtverec má osm sousedů, se čtyřmi z nich sdílí právě jednu hranu a se čtyřmi dalšími, diagonálními sousedy sdílí právě jeden bod. To zavádí jedno z hlavních omezení čtvercové sítě. Otočení objektu je možné pouze o násobky pravého úhlu, je-li uvažován přechod do sousedních buněk jedině ve směru tečném na hranu, aby všechny sousední stavy byly stejně vzdáleny. Vzdáleností stavů je myšlena délka

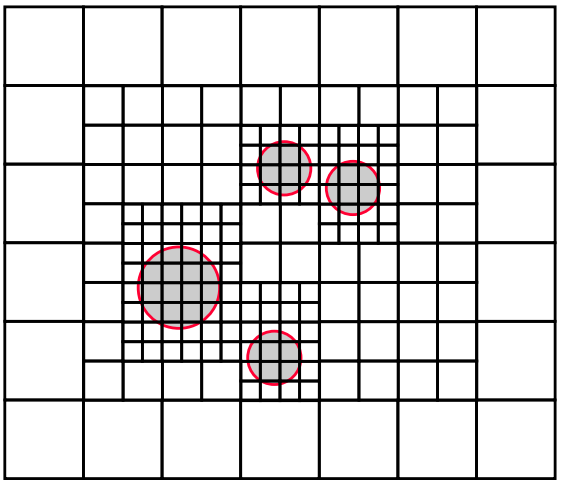


(a) Čtvercová mřížka (b) Šestihranná mřížka

Obrázek 2.1: Navigace pomocí mřížky

úsečky spojující středy stavů. Tento přístup znázorňuje modrá cesta na obrázku 2.1(a). Otočení o násobky 45° je možné, je-li navíc umožněn přechod do buněk sousedících na diagonálách, viz žlutá cesta na obrázku 2.1(a).

Oproti tomu všechny sousední stavy šestiúhelníku, se kterými sdílí hranu, mají stejnou vzdálenost. Dalších šesti sousedů lze dosáhnout přechodem přes vrchol. Změna směru je proto možná o násobky 60°, jak znázorňuje modrá cesta na obrázku 2.1(b), nebo o násobky 30° v případě žluté cesty na tomtéž obrázku. Další výhodou je i efektivnější pokrytí členité plochy, jak je zřejmé z porovnání na obrázku 2.1. Na popis stejné plochy bylo použito 35 čtvercových buněk, ale 28 šestiúhelníkových buněk srovnatelné velikosti. Nalezená cesta v šestiúhelníkové síti vypadá přirozeněji, viz obrázek 2.1(b). Nevýhodou této metody jsou náročnější operace s mřížkou.



Obrázek 2.2: Adaptivní segmentace

Důležitou vlastností mřížky je v obou případech velikost buněk, která musí být vhodně zvolena s ohledem na velikost objektů, jejichž pohyb bude plánován. Například

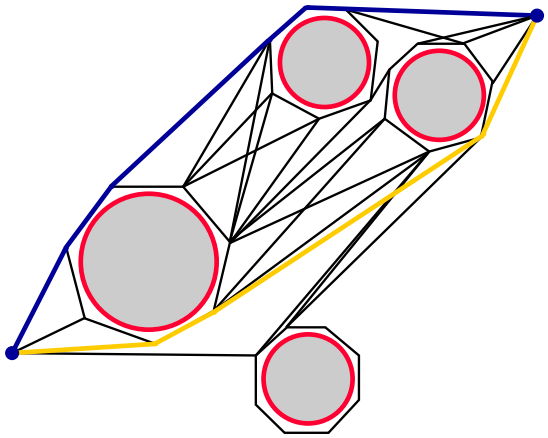
při plánování pohybu osob je nejlepší vycházet z šířky ramen, aby naváděný člověk mohl bez potíží projít všechny nalezené cesty.

Ve speciálních případech, kdy se jedná o rozlehlejší mapu s větším volným prostorem je možné použít adaptivní velikost buněk. To znamená, že volný prostor je popsán velkými buňkami a jak se objekt přibližuje k překážce, velikost buněk se dynamicky zmenšuje a zpřesňuje se tak plánování v blízkosti překážky. Tato metoda je popsána v disertační práci D. Šišláka [9]. Příklad prostoru pokrytého adaptivní sítí je znázorněn na obrázku 2.2. Adaptivní segmentace ušetří čas potřebný k prohledávání, protože algoritmus nebude prohledávat detailně popsané prázdné místo, jako by tomu bylo v případě použití pevné velikosti mřížky.

2.2.1.2 Polygonální mapy

Popis prostoru pomocí polygonálních map je vhodný především pro velké prostory homogenního charakteru s minimem překážek. Na rozdíl od předchozí metody, polygonální mapy pracují s grafovou reprezentací prostoru. Všechny překážky jsou obaleny polygony, jejichž vrcholy tvoří stavový prostor. Graf viditelnosti je generován propojením vzájemně viditelných vrcholů polygonů. Příklad prostoru popsaného pomocí polygonální mapy je znázorněn na obrázku 2.3.

Protože stavový prostor je tvořen pouze polygony které popisují překážky, při nízkém počtu překážek je tato metoda výhodná pro svou paměťovou nenáročnost a možnost rychlého prohledání. Naopak nevýhodou je, že objekt je naváděn prostorem v těsné blízkosti překážek. Protože viditelné stavy jsou propojovány přes celý volný prostor, není možné zohlednit lokální vlastnosti prostředí.



Obrázek 2.3: Navigace podle polygonální mapy

2.2.1.3 Navigační sítě

Principem metody popisu virtuálního světa pomocí navigačních sítí je, že volný prostor je popsán n-úhelníky a stavový prostor je tvořen středy těchto n-úhelníků, nebo středy jejich stran. Tímto způsobem je možné nejen snížit paměťové nároky na reprezentaci velkých prostorů, ale také rozdělit virtuální svět do logických bloků. S logickými bloky je pak možné spojovat nejrůznější informace, které mohou ovlivnit plánování cesty. Například označení místa přepadení při použití metody v počítačových hrách, může znevýhodnit kratší cestu, takže bude vybrána delší, zato však bezpečnější cesta [8]. Ilustrativní příklad popisu prostoru pomocí navigačních sítí je znázorněn na obrázku 2.4.

V článku autorů D. Hale a kolektiv [7] jsou porovnávány tři algoritmy pro generování navigačních sítí, algoritmus DEACCON (*Decomposition of Environments for the Creation of Convex-region Navigation-meshes*), Hertel-Mehlhornův algoritmus a algoritmus SFV (*Space Filling Volumes*).

Algoritmus SFV používá k popisu volného prostoru čtyřúhelníků. Čtyřúhelníky se zvětšují, dokud nedosáhnou překážky, pak je růst v tomto směru zastaven. Nevýhodou tohoto algoritmu je, že pomocí čtyřúhelníků není možné popsat volný prostor beze zbytku, pokud překážky nejsou osově zarovnané.

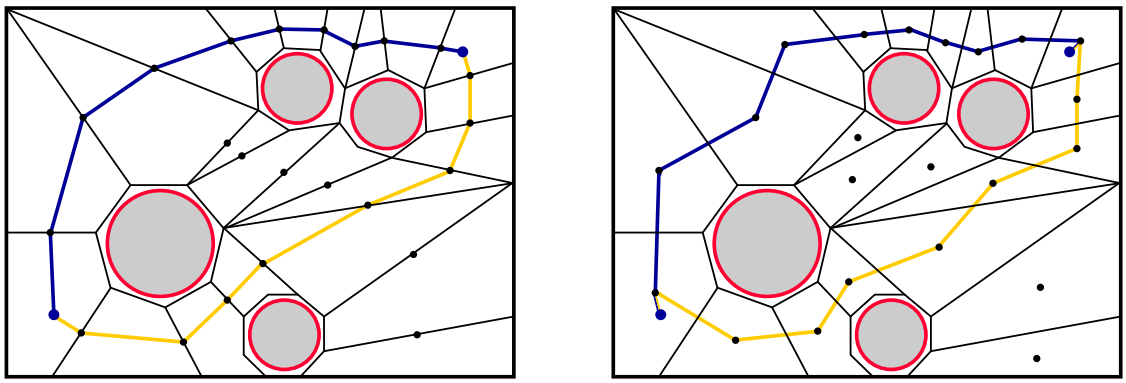
Hertel-Mehlhornův algoritmus používá k popisu prostoru trojúhelníkovou dekompozici. Pomocí této metody je možné beze zbytku popsat jakýkoli prostor. Nevýhodou této metody je, že volný prostor může být algoritmem rozložen na velký počet velice úzkých trojúhelníků.

Algoritmus DEACCON, který je představen v článku [7], k popisu prostoru využívá podobných principů jako algoritmus SFV. Hlavním rozdílem ale je, že po dosažení překážky algoritmus otestuje zda je hrana překážky rovnoběžná s hranou n-úhelníku. V případě rovnoběžnosti obou hran je růst n-úhelníku v daném směru zastaven. Pokud hrany nejsou rovnoběžné, znamená to že je možné v daném směru dále expandovat, pokud bude do n-úhelníku přidána hrana rovnoběžná s hranou překážky. Aby byla zaručena konvexita n-úhelníků, je po každém cyklu růstu proveden test viditelnosti vrcholů n-úhelníku. Nové n-úhelníky jsou stále přidávány na místa, která nelze popsat již existujícími n-úhelníky z důvodu zachování konvexity, dokud není bezezbytku popsán celý volný prostor.

Na obrázku A.1 v příloze dokumentu je zobrazeno porovnání výše zmíněných algoritmů pro popis prostředí pomocí navigačních sítí.

2.2.2 Prohledávací algoritmy

Po vytvoření stavového prostoru lze použít některý z prohledávacích algoritmů k nalezení cesty. Existuje celá řada způsobů prohledávání. Základ tvoří takzvané neinformo-



(a) Stavový prostor tvoří středy stran. (b) Stavový prostor tvoří středy polygonů.

Obrázek 2.4: Navigace podle navigační sítě

vané algoritmy, neinformované proto, že k nalezení cesty nevyužívají žádné informace o prostředí, ani o řešeném problému. Pokročilejší, informované algoritmy naopak využívají informace o prostředí a řešené úloze a prohledávání tak co nejvíce optimalizují. Na rozdíl od neinformovaných algoritmů jsou schopné najít optimální řešení v kratším čase a s menším počtem expandovaných uzlů. Expanzí uzlu je myšleno nalezení bezprostředně sousedících uzlů. Samozřejmě lze využít i specializovaných algoritmů, které jsou vyvinuty pro řešení konkrétních problémů. Příkladem je dynamika prostředí, kde se užívá takzvaných *lifelong* plánovacích algoritmů, nebo přirozenost nalezené cesty, kterou řeší *any-angle* plánovací algoritmy.

Popis následujících algoritmů vychází především z publikace Mařík a kolektiv [15], disertační práce D. Šišláka [9] a publikace [10].

2.2.2.1 Neinformované prohledávací algoritmy

Jak již bylo řečeno, neinformované metody jsou základní možností prohledávání. Jejich význam spočívá v tom, že mnoho pokročilejších algoritmů je odvozeno, nebo inspirováno právě neinformovanými metodami. Spíše než pro praktické použití se tyto algoritmy hodí k přiblížení problematiky prohledávání stavového prostoru. Hlavní nevýhodou neinformovaných algoritmů je to, že pro nalezení cesty prohledávají nadbytečně velký prostor. Pro prohledávání využívají seznamů OPEN a CLOSED. V seznamu OPEN se uchovávají uzly, které ještě nebyly expandovány. Seznam CLOSED slouží pro uchování již expandovaných uzlů, je nutné jej použít v případě, kdy stavový prostor obsahuje cykly. Hlavní rozdíl následujících algoritmů je ve strategii, jakou používají pro výběr expandovaného uzlu.

BFS (Breadth First Search)

Jedná se o algoritmus slepého prohledávání do šířky. To znamená, že přednostně jsou expandovány uzly s nejmenší hloubkou. Hloubka uzlu ve stromu je rovna počtu hran od kořene k danému uzlu. Paměťová i časová náročnost algoritmu je $O(b^d)$, kde b je maximální faktor větvení a d je hloubka cílového uzlu. Právě kvůli paměťové náročnosti je tento algoritmus nevhodný pro prohledávání rozsáhlých stavových prostorů. Algoritmus je úplný a optimální, to znamená že existuje-li řešení, pak bude nalezeno to v nejmenší hloubce a algoritmus skončí.

DFS (Depth First Search)

Druhým neinformovaným algoritmem je slepé prohledávání do hloubky. U tohoto algoritmu jsou naopak přednostně expandovány uzly s největší hloubkou. Algoritmus DFS není úplný, ani optimální. A to ani pro konečný stavový prostor, protože v něm mohou existovat smyčky, ve kterých algoritmus DFS uvázne. Tento problém řeší algoritmus s pevnou hloubkou prohledávání (DLDFS), kde je hloubka prohledávání pevně omezena, nebo algoritmus s proměnnou hloubkou prohledávání (IDDFS), u kterého je hloubka prohledávání v každém kroku dynamicky zvětšována o 1, dokud není nalezeno řešení. Tyto variace algoritmu DFS jsou úplné a optimální, v případě DLDFS musí být splněna podmínka, že hloubka cílového uzlu je menší, než limit prohledávání. Paměťová náročnost je $O(bd)$. Časová náročnost je stejně jako u BFS algoritmu $O(b^d)$.

2.2.2.2 Informované prohledávací algoritmy

Informace o úloze a jejím možném řešení jsou využívány k omezení nadbytečného větvení stavů a k soustředění prohledávání směrem k cíli. Tím je významně urychleno prohledávání a také dojde ke snížení paměťové náročnosti algoritmu. Zásadní informací o řešené úloze je znalost cílového stavu. V mnoha aplikacích je výhodné definovat hodnotící funkci, která přiřadí každému stavu jednu hodnotu, například v závislosti na ceně, spotřebě, vzdálenosti, bezpečnosti a dalších kritériích. Ohodnocení je následně využíváno prohledávacím algoritmem pro výběr expandovaného uzlu a soustředění prohledávání směrem k cílovému stavu.

Gradientní algoritmus

Jedná se o nejjednodušší informovaný algoritmus pro prohledávání stavového prostoru. Vždy je expandován ten uzel, který má nejlepší ohodnocení a z následníků je pro další expanzi vybrán opět nejlépe ohodnocený uzel. Všechny ostatní uzly jsou zapomenuty. Prohledávání je zastaveno, když je expandován uzel, jehož následníci mají horší ohodnocení, než on sám. Nevýhodou tohoto algoritmu je, že může uváznout v lokálním

extrému, globálního extrému tak nemusí dosáhnout. A protože není uchována historie, musí se restartovat a začít prohledávání znovu z náhodného místa. Naopak výhodou je rychlost a paměťová nenáročnost.

Algoritmus uspořádaného prohledávání

Tento algoritmus vznikl rozšířením Gradientního algoritmu o paměť, tedy seznamy OPEN a CLOSED. Na rozdíl od neinformovaných algoritmů uváděných v předchozí kapitole se ve zmiňovaných seznamech ukládá více informací. Označení uzlu zůstává, ale je přidána ještě hodnota uzlu (určená hodnotící funkcí) a rodičovský uzel. Na základě těchto informací pak lze při uváznutí pokračovat v hledání z jiného uzlu na cestě. V případě nalezení cílového uzlu lze pomocí rodičovských referencí sestrojit cestu.

Simulované žíhání

Simulované žíhání je další modifikací gradientního algoritmu. Tento algoritmus je inspirován procesem žíhání, při němž se materiál (například ocel), zahřeje na žíhací teplotu a následně se nechá řízeně ochlazovat. Díky změnám ve struktuře materiálu dojde ke zlepšení některých vlastností. Podle této analogie je před prvním krokem algoritmu pomyslná teplota T nastavena na určitou hodnotu, která se postupem času snižuje. Pravděpodobnost P expanze uzlu s' z množiny následníků uzlu s je funkcí teploty T a pomyslných energií $e = E_{(s)}$ a $e' = E_{(s')}$, tedy $P = f_{(T,e,e')}$. Energie uzlů mohou být reprezentovány různými veličinami, které se v závislosti na konkrétní aplikaci snažíme minimalizovat. Například při hledání nejkratší cesty budou energie reprezentovány vzdálenostmi. Díky pravděpodobnostnímu přístupu mohou být expandovány i uzly s horším ohodnocením, což zabraňuje uváznutí v lokálním minimu. S klesající teplotou klesá i pravděpodobnost přijetí horšího řešení a algoritmus se stále více chová jako gradientní. Chování algoritmu tedy převážně závisí na rychlosti konvergence teploty a také na tom, jak je definována funkce $P = f_{(T,e,e')}$, která určuje pravděpodobnost expanze daného uzlu.

Tento algoritmus popisují autoři D. Bertsimas a J. Tsitsiklis v článku [3].

Dijkstrův algoritmus

Dijkstrův algoritmus je grafový algoritmus, který slouží k nalezení nejkratší cesty v kladně ohodnoceném grafu G, kde V je množina všech vrcholů a E je množina všech hran grafu. Algoritmus pro každý vrchol v' ukládá nejmenší cenu d s jakou jej lze dosáhnout a rodičovský vrchol v. Cena vrcholu v' je rovna součtu ceny, za jakou lze dosáhnout rodičovského vrcholu $d_{(v)}$ a ceny přechodu mezi rodičem a potomkem $l_{(v,v')}$. Nejkratší cesta je po ukončení prohledávání sestrojena pomocí rodičovských referencí. Časová

náročnost algoritmu je závislá na implementaci, jeli použito pole pro uložení vrcholů, pak je paměťová náročnost algoritmu dána závislostí $O(|V|^2 + |E|)$, kde $|V|$ je počet vrcholů a $|E|$ je počet hran grafu.

Algoritmus A*

K soustředění prohledávání využívá hodnotící funkci $f^*_{(s)} = g^*_{(s)} + h^*_{(s)}$. Kde $f^*_{(s)}$ je odhad celkové ceny cesty přes daný uzel s, $g^*_{(s)}$ je odhad ceny cesty do uzlu s a $h^*_{(s)}$ je odhad ceny cesty z uzlu s do cílového uzlu. V oblasti plánování pohybu je mnohdy možné určit cenu cest do daného uzlu přesně. Tedy místo odhadu $g^*_{(s)}$ lze použít přesný údaj $g_{(s)}$. Heuristická funkce $h^*_{(s)}$ musí být přípustná, to znamená že odhad ceny musí být větší, nebo roven nule a nejvýše roven skutečné ceně cesty. Jestliže je tato podmínka splněna, pak algoritmus A* nalezne optimální cestu. Čím více se heuristická funkce $h^*_{(s)}$ blíží skutečné funkci $h_{(s)}$, tím menší část stavového prostoru je prohledávána. Algoritmus pro svoji funkci využívá seznamů OPEN a CLOSED. Expandován je vždy ten uzel ze seznamu OPEN, který má nejnižší ohodnocení $f^*_{(s)}$. Po expanzi je uzel přesunut do seznamu CLOSED a všichni jeho následníci jsou připsáni do seznamu OPEN. Nalezená cesta je po ukončení prohledávání sestrojena pomocí rodičovských referencí.

Algoritmus A* je jedním z nejužívanějších algoritmů a vychází z něho mnoho dalších. Jednou z variant je algoritmus IDA*. Funkce je založena na podobném principu, jako je tomu u neinformovaného algoritmu IDDFS, který je popsaný v předešlé kapitole. Místo omezení hloubky se dynamicky zvyšuje povolená hodnota funkce $f^*_{(s)}$. Na počátku je omezení stanoveno na odhad ceny cesty mezi počátečním a koncovým uzlem. Pokud by cenou byla myšlena vzdálenost uzlů, tak na počátku bude funkce $f^*_{(s)}$ nastavena na délku úsečky spojující počáteční a cílový uzel. Na konci každého kroku se pak hodnota $f^*_{(s)}$ navyšuje o minimální hodnotu, o kterou byla v tomto kroku překročena. Zavedením tohoto omezení se výrazně sníží nároky na paměť, protože nejsou uvažovány ty uzly, jejichž cena přesáhla prahovou hodnotu $f^*_{(s)}$.

2.2.2.3 Speciální prohledávací algoritmy

V celé řadě aplikací je nutné řešit problémy, které kladou vyšší nároky na plánovací algoritmus. Příkladem je časté přeplánování cesty v dynamickém prostředí. Pokud nedochází ke změnám často, je možné provést nové plánování s konvenčním prohledávacím algoritmem. Od určité četnosti změn ale požadavky na výkon potřebný k neustálému přeplánování vzrostou nad únosnou míru. Například při exploraci neznámého terénu mobilním robotem. Proto byly vyvinuty takzvané *lifelong* plánovací algoritmy tak, aby pro přeplánování cesty využily informace z předchozího prohledávání a tím minimalizovaly výpočty spojené se změnami v prostředí. Do skupiny lifelong algoritmů patří například

*Lifelong planning A**, nebo *D* Lite*. *Lifelong* algoritmy se zabývají autoři S. Koenig a kolektiv v publikacích [11] a [12], ze kterých vychází následující podkapitola.

Druhým případem je plánování cest v prostředí, kde je třeba provádět vyhlazování již naplánované cesty dalším algoritmem. Příkladem mohou být prostředí popsaná navigačními mřížkami. *Any-angle* algoritmy, česky bychom mohli říci všesměrové algoritmy, řeší přirozenost cesty už při jejím plánování. Tato vlastnost je zvláště užitečná v prostorech s větším počtem dimenzí. *Any-angle* prohledávací algoritmy jsou například *Field D**, nebo *Theta**. Podrobnější informace včetně pseudokódů obou algoritmů lze nalézt v publikaci [18]. Dalším podkladem pro podkapitolu o *any-angle* plánovacích algoritmech byl online článek [19].

Tyto algoritmy jsou využitelné především pro potřeby plánování pohybu, jejich základem je prohledávací algoritmus A*. Vzhledem ke složitosti a vzájemné podobnosti algoritmů bude v této kapitole popsán pouze jeden z každé skupiny.

Lifelong planning A*

Tento algoritmus se používá pro prohledávání stavového prostoru, který reprezentuje dynamické prostředí v němž se často mění cena přechodů mezi stavy. Algoritmus LPA* je rozšířením algoritmu A*, který je popsán v kapitole 2.2.2.2. Definice funkcí $g^*_{(s)}$ a $h^*_{(s)}$ je totožná jako u algoritmu A*, navíc je definována ještě pomocná funkce rhs. Ta je obdobou funkce g^*. Pro počáteční uzel s_{START} je tato funkce definována pomocí rovnice (2.1), pro všechny ostatní uzly je funkce rhs definována pomocí rovnice (2.2), kde $Pred(s)$ reprezentuje množinu předchůdců uzlu s a $c_{(s',s)}$ je cena přechodu z uzlu s' do uzlu s. Hodnota rhs je ohlédnutím o jeden krok zpět, slouží k porovnání zda nelze uzlu dosáhnout levněji.

$$rhs_{(s_{START})} = 0 \tag{2.1}$$

$$rhs_{(s)} = \min_{s' \in Pred(s)} (g_{(s')} + c_{(s',s)}) \tag{2.2}$$

Uzel s je lokálně konzistentní právě tehdy, když $g^*_{(s)} = rhs_{(s)}$. Jsou-li všechny uzly lokálně konzistentní, pak lze nalézt nejkratší cestu z uzlu s_{START} do jakéhokoli uzlu s_{GOAL}. Nejkratší cestu lze najít tak, že počínaje uzlem $s = s_{GOAL}$ přecházíme vždy do uzlu $s' \in Pred(s)$, který má minimální hodnotou $g_{(s')} + c_{(s',s)}$.

Dojde-li ve stavovém prostoru ke změnám, LPA* zařadí nekonzistentní uzly do fronty a na rozdíl od klasického plánování přepočítá pouze ty, které jsou důležité pro naplánování cesty mezi s_{START} a s_{GOAL}. Nekonzistentní uzly ve frontě jsou řazeny podle klíče, definice viz rovnice 2.3. První hodnota klíče (2.3) odpovídá hodnotě $f^*_{(s)}$ algoritmu A*, druhá hodnota klíče odpovídá $g^*_{(s)}$ algoritmu A*.

$$\begin{aligned} K_{(s)} &= \{K_1, K_2\} \\ K_1 &= \min(g^*_{(s)}, rhs_{(s)} + h^*_{(s)}) \\ K_2 &= \min(g^*_{(s)}, rhs_s) \end{aligned} \tag{2.3}$$

Při inicializaci jsou g^* hodnoty všech uzlů nastaveny na nekonečno a rhs hodnoty podle definice, viz (2.1) a (2.2). Jediný nekonzistentní uzel je startovní uzel. Z popisu výše vyplývá, že při prvním průběhu algoritmus LPA* provede přesně totéž, co algoritmus A*. Po nalezení cesty algoritmus čeká na změny v prostředí.

Algoritmus Theta*

Algoritmus Theta* má dvě varianty, klasickou a angle-propagation. Na rozdíl od algoritmu A*, Theta* umožňuje aby rodičem uzlu byl jiný, než sousední uzel. V důsledku toho lze najít takovou cestu, která již nepotřebuje vyhlazování.

Základní algoritmus Theta* se při expanzi řídí stejnými pravidly, jako algoritmus A* popsaný v kapitole 2.2.2.2. Algoritmus expanduje vždy uzel s, který má nejmenší hodnotu $f^*_{(s)}$. Poté je zkontrolována přímka viditelnosti na prarodiče uzlu s. Je-li prarodič viditelný, pak je nastaven jako rodič a je upravena hodnota $g_{(s)} = g_{(s'')} + c_{(s'',s)}$, kde $c_{(s'',s)}$ je cena přechodu z prarodiče s'' do uzlu s. Nemá-li uzel s přímku viditelnosti na prarodiče, pak Theta* funguje jako A*, tedy nastaví jako rodiče uzel s', který přímo předchází a hodnotu $g_{(s)} = g_{(s')} + c_{(s',s)}$. Výsledné cesty vypadají realisticky a jsou krátké. Nalezení nejkratší možné cesty ale nelze zaručit. Hlavní nevýhodou tohoto algoritmu je, že musí provádět mnoho testů viditelnosti, které jej zpomalují.

AP Theta* eliminuje potřebu testování viditelnosti tím, že při expanzi uzlu jsou stanoveny mezní úhly, $ub_{(s)}$ a $lb_{(s)}$. Mezní úhly uzlu mohou být stanoveny například pomocí viditelných překážek. Úhel Θ je definován následujícím vztahem $\Theta = \angle(s, p, s')$, kde s' je následník a p je rodič uzlu s. Úhel Θ je kladný, leží-li úsečka $|ps'|$ proti směru hodinových ručiček od úsečky $|ps|$. Jakmile se úhel následníka uzlu s nachází v rozmezí úhlů $ub_{(s)}$ a $lb_{(s)}$, je garantováno, že rodič a následník uzlu s jsou navzájem viditelní. Rodičovské reference i hodnota g jsou pak nastaveny jako u klasické verze algoritmu Theta*.

2.3 Plánování trajektorií

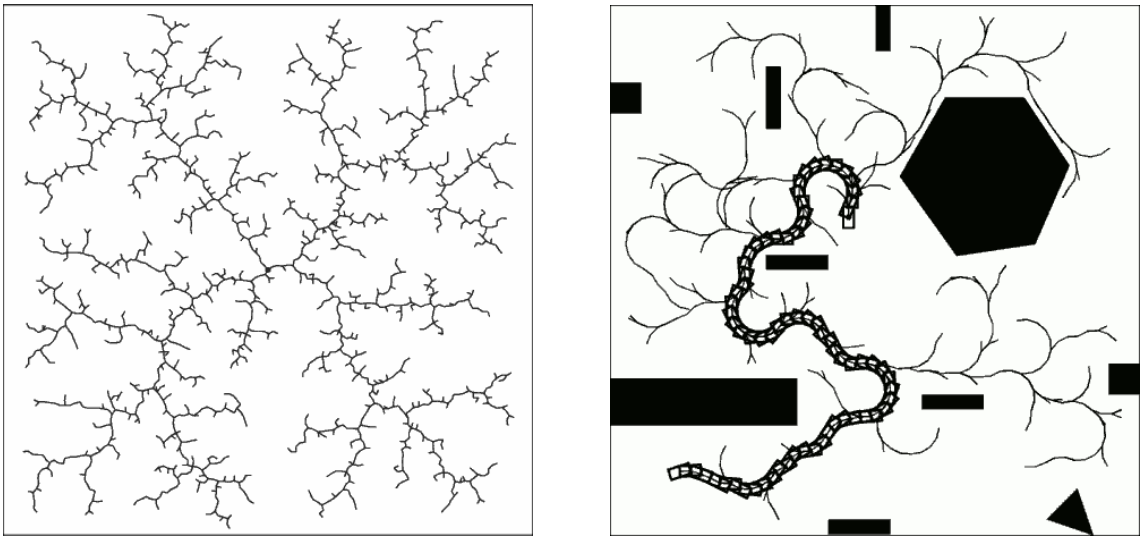
Při řešení komplexnějších problémů lze plánovat přímo trajektorii pohybu objektu. K tomuto problému je možné přistupovat jako k optimalizační úloze a pro nalezení optimální trajektorie využít například genetických algoritmů ve spojení s algoritmem RRT. Příkladem takovéto aplikace je detailní plánování pohybu letadel, kdy je plánování

zásadním způsobem ovlivňováno kinematickými a dynamickými vlastnostmi letadla.

2.3.1 Rapidly-Exploring Random Trees

Jedná se o algoritmus, který je používán především pro plánování složitých trajektorií ve 3D prostředí s uvažováním kinematických a dynamických omezení objektu, lze jej ale použít i pro plánování cesty. Díky rychlosti algoritmu jsou stromy generovány při běhu plánovacího algoritmu, tedy pro každý další běh je vytvořen nový prohledávací strom. Proto je pro každý průběh algoritmu nalezena i jiná trajektorie a nelze tedy zaručit nalezení optimální trajektorie.

Vstupem funkce pro vytvoření stromu je počáteční stav objektu x_{init} a časový interval Δt. Počáteční stav je přidán do stromu τ jako první a poté se spustí cyklus, v každé iteraci je nejprve vygenerován náhodný stav x_{rand}. Dalším krokem je nalezení nejbližšího souseda stavu x_{rand} z již známých stavů v generovaném stromu τ, označovaného x_{near}. Následně je v závislosti na dynamice objektu, délce časového intervalu Δt a poloze blízkého uzlu x_{near} určen stav x_{new} tak, aby jej naváděný objekt byl schopen za daných podmínek dosáhnout. Na konci každé iterace je do stromu τ přidán uzel x_{new} a hrana, která jej spojuje s x_{near}. Cyklus je ukončen, když je ke stromu připojen cílový stav. Po ukončení je navrácen vygenerovaný strom, v němž lze nalézt trajektorii pohybu. V případě plánování cesty tímto algoritmem jsou nové body připojovány k existujícímu stromu pouze úsečkou definované délky.



(a) Rapidní explorace volného prostoru. Tento obrázek byl převzat z [5].

(b) Nalezení trajektorie objektu. Tento obrázek byl převzat z [5].

Obrázek 2.5: Příklad použití algoritmu RRT

Na obrázku 2.5(a) je znázorněn způsob, jakým je popisován volný prostor při generování stromu algoritmem RRT. Obrázek 2.5(b) demonstruje použití algoritmu RRT pro

nalezení trajektorie objektu s uvažováním kinematických omezení.

Tato podkapitola vychází z práce S. M. LaValle [13] a online článku [5]. Další informace je možné nalézt také v publikaci S. M. LaValle [14].

2.3.2 Genetické algoritmy

Genetické algoritmy jsou využívány především ve složitých, vícedimenzionálních úlohách, kde do plánování pohybu objektu vstupuje i jeho kinematika a dynamika. Pro plánování pohybu osob se genetické algoritmy příliš nehodí.

Tato podkapitola vychází z disertační práce D. Šišláka [9], publikace autorů Mařík a kolektiv [16] a článku [6].

Genetické algoritmy jsou inspirovány přírodními principy, konkrétně vývojem živých organizmů. Optimální řešení úlohy se vyvíjí ze série, na počátku náhodně vygenerovaných řešení. Chromozomy, tedy zmíněná dílčí řešení, jsou interpretovány v podobě stejně dlouhých řetězců složených ze znaků definované abecedy. Podle toho, jak dobře chromozomy řeší daný problém, je každému přiřazena takzvaná fitness hodnota. Množina ohodnocených chromozomů se nazývá generace. Funkce genetických algoritmů je založena na několika základních operacích, které jsou na chromozomech dané generace vykonávány s určitou pravděpodobností. Jednou z nejdůležitějších operací je selekce, jejímž cílem je vybrat chromozomy, které budou tvořit další generaci. Míra jeho pravděpodobnosti, že bude chromozom vybrán do dalšího kroku algoritmu je úměrná fitness hodnotě. Dalším, neméně důležitým mechanizmem je křížení. Všechny chromozomy jedné generace, mají stejnou pravděpodobnost křížení, řádově jednotky procent. Křížení páru chromozomů probíhá tak, že je náhodně určen bod křížení, tedy přirozené číslo v intervalu $(0, n)$, kde n je délka řetězce. V tomto bodě jsou obě řešení přerušena a dva chromozomy další generace vzniknou spojením levé části prvního řetězce s pravou částí druhého řetězce a naopak. Mutace je dalším důležitým mechanizmem vývoje řešení, také nastává se stejnou pravděpodobností pro všechny chromozomy dané generace, pravděpodobnost je řádově desetiny procent. Mutace vybraného chromozomu probíhá tak, že je určen index znaku, opět je to přirozené číslo v intervalu $(0, n)$. Znak na tomto místě je pak nahrazen jiným, náhodně vybraným znakem abecedy a zmutovaný chromozom je zařazen do následující generace. Vzhledem k náhodnému přístupu se může stát, že dosud nejlepší chromozom bude zapomenut, a proto je nutné nejlepší řešení uchovávat mimo generace.

u_{s_1}	u_{Φ_1}	x_1	y_1	θ_1	b_1	...	u_{s_n}	u_{Φ_n}	x_n	y_n	θ_n	b_n

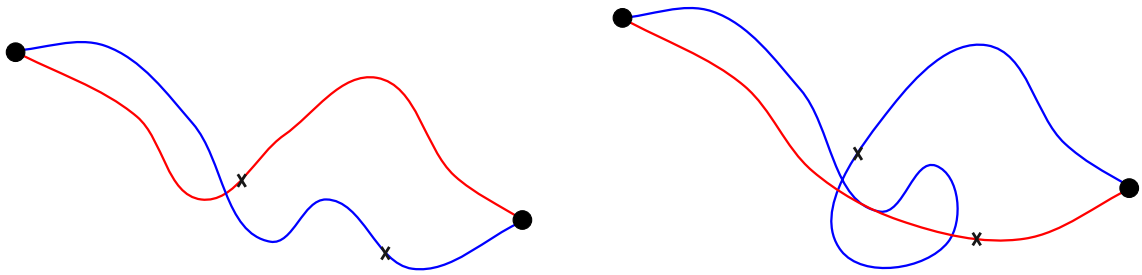
Obrázek 2.6: Chromozom

V článku [6] autoři zabývají plánováním trajektorií objektu se složitou dynamikou, pomocí genetických algoritmů. Matematický popis složitějšího objektu znázorňuje sou-

stava rovnic 2.4, v tomto případě se jedná popis automobilu. Problematiku kinematiky objektu detailně popisuje S. M. LaValle v publikaci [14]. Chromozom je v článku [6] definován jako posloupnost rychlostí (u_{s_i} a u_{Φ_i}), souřadnic (x_i a y_i), úhlů (θ_i) a logických proměnných (b_i), které reprezentují dosažitelnost uzlu. Je-li některý uzel nedosažitelný, pak je celá trajektorie považována za nerealizovatelnou. Příklad reprezentace chromozomu je na obrázku 2.6. Na rozdíl od teoretického popisu genetických algoritmů, chromozomy v této aplikaci mohou mít rozdílnou délku.

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{y} &= u_s \sin\theta \\ \dot{\theta} &= \frac{u_s}{L}\tan u_\phi \end{aligned} \tag{2.4}$$

Počáteční populace je generována pomocí RRT algoritmu, popsaného v kapitole 2.3.1. Tento algoritmus je ale pro urychlení činnosti upraven tak, že neověřuje bezkolizní trajektorii a pouze aplikuje dynamická omezení objektu. Trajektorie z počáteční generace proto mohou vést i přes překážky. Fitness funkce chromozomu je definována jako převrácená hodnota ceny dané trajektorie, více podrobností je uvedeno v kapitole D článku [6].



(a) Trajektorie před křížením (b) Trajektorie po křížení

Obrázek 2.7: Křížení chromozomů

Při křížení dvou chromozomů v této konkrétní aplikaci jsou také vybrány dva chromozomy, oba jsou ale přerušeny v náhodných místech a poté rekombinovány. Aby nebyly porušeny požadavky kladené dynamikou objektu, k rekombinaci, tedy spojení dvou částí nového chromozomu dochází pomocí RRT algoritmu. Příklad křížení chromozomů je znázorněn na obrázku 2.7.

Mutace chromozomů má v této aplikaci čtyři podoby. První typ mutace je aplikovatelný jak na realizovatelné, tak na nerealizovatelné chromozomy. Cílem tohoto druhu mutace je optimalizace trajektorie. Princip spočívá v tom, že je vybrán náhodný gen. Z jeho okolí o poloměru ϵ je náhodně vybrán druhý gen. Následně je trajektorie mezi nimi nahrazena novým propojením pomocí RRT algoritmu. Druhý typ mutace je téměř shodný s prvním typem, ovšem s tím rozdílem, že zde je druhý bod mutace vybrán z celého dostupného prostoru. Poslední dva typy mutace jsou aplikovatelné pouze na ne-

realizovatelné trajektorie, jejich cílem je odstranění částí trajektorie, které vedou přes překážky. Více o mutacích se můžete dočíst v kapitole C článku [6].

Proces evoluce je shodný s teoretickým procesem popsaným výše. Kvalita nalezené trajektorie je závislá na počtu kroků, které má evoluční proces. Nejlepším přístupem je určit podmínku ukončení procesu a po jejím splnění nechat proces iterovat definovaný počet kroků. Podmínkou ukončení je nalezení první realizovatelné trajektorie. Účelem přidání určitého počtu kroků je optimalizace nalezené trajektorie.

3 Plánovací algoritmy a jejich omezení

3.1 Překážky ve virtuálním prostředí

Překážkou nezbytně nemusí být cizí objekt stojící v cestě, ale může jí být i příliš prudké stoupání, úzký průchod, hustý les a mnohé další. Ve virtuálních aplikacích lze těchto překážek využít k ohraničení prostoru, případně k oddělení některých částí mezi kterými je umožněno procházet pouze definovanými průchody. Tohoto přístupu se hojně využívá v počítačových hrách, kde je nutné hráče omezit v pohybu pouze na určitou herní mapu.

Jak jsou překážky ve virtuálním prostředí reprezentovány je popsáno v předchozí kapitole. V některých případech je vhodné před vlastní diskretizací provést takzvané ,,nafukování"překážek. Principem této metody je, že se překážka zvětší o hodnotu Δx směrem od jejího středu. Překážka se tedy ,,nafoukne", aby plánování pohybu objektů mohlo probíhat bez kontroly kolizí. Hodnotu, o kterou budou překážky zvětšeny je vhodné zvolit s ohledem na velikost objektů, jejichž pohyb bude plánován.

Výše byly popsány pouze statické překážky, jejichž poloha se s časem nemění. V mnoha aplikacích se však překážky mohou pohybovat, objevovat se a zase mizet. Například rozšiřující se oheň, nebo dveře, při simulaci evakuace hořícího domu může reprezentovat dynamickou překážku. V počítačových hrách je to například brána kterou lze zavírat, nebo padací most. Pokud se prostředí často mění lze s výhodou zvolit některý z *lifelong* plánovacích algoritmů. Nevýhodou těchto algoritmů ale je, že pro správnou funkci je nutné uchovávat v paměti kopii stavového prostoru, což by při plánování pohybu velkého počtu objektů mohlo způsobit potíže.

V některých aplikacích se mohou vyskytnout situace, kdy je třeba plánovat pohyb objektů ve více úrovních mapy nad sebou. Takovéto prostředí se označuje jako 2.5D. Příkladem může být například vícepatrová budova, podchod pod silnicí, nebo tunel. Překážky jsou v tomto prostředí, jak již bylo řečeno v úvodu této podkapitoly, používány k ohraničení povoleného prostoru, například v podobě zdí. Pokud se jedná o rozsáhlejší prostor, jako je například patro budovy, je vhodné diskretizovat každou z úrovní mapy zvlášť a po diskretizaci ve stavovém prostoru definovat průchody mezi jednotlivými úrovněmi prostředí.

3.2 Kinematická a dynamická omezení pohybu objektů

Významnou roli při plánování pohybu hraje kinematika a dynamika objektu, tedy jistá fyzikální omezení, která musí objekt splňovat, aby jeho pohyb byl přirozený. S rostoucí náročností popisu stoupají i požadavky na plánovací algoritmus. Podle tohoto kritéria je možné plánování rozdělit do tří skupin, a to na: holonomní, neholonomní a

kinodynamické.

Dynamika je obor fyziky, který se zabývá příčinami pohybu a popisuje jej z hlediska působení sil. Oproti tomu kinematika, se zabývá spíše klasifikací pohybu. Kinematikou objektu je myšlen popis objektu na základě jeho kinematických vlastností, jako například okamžitá rychlost, případně úhel natočení kol a další. Pod pojmem dynamika objektu je myšlen popis závislosti již zmíněných kinematických veličin a jejich vývoj v čase.

3.2.1 Holonomní plánování

Pohyb objektu je holonomní, je-li možné vztah mezi jednotlivými body trajektorie popsat jako funkci jejich polohy a času. Spíše než o plánování trajektorie se v tomto případě ale hovoří o plánování cesty, která je plánována nezávisle na čase. Z hlediska plánování se jedná o nejjednodušší úlohu ze tří jmenovaných. Objekt není v pohybu nijak kinematicky ani dynamicky omezen, nebo jsou tato omezení zanedbána. Jejich zanedbáním ale dojde k tomu, že výsledná trajektorie objektu je nepřirozená.

Za objekt, jehož trajektorie je holonomní, může být považován člověk. Protože jeho trajektorie je minimálně kinematicky a dynamicky omezená, zanedbáním těchto omezení nevznikne téměř žádná chyba. Zvláště pokud se naváděná osoba pohybuje malou rychlostí. Člověk na procházce může bez větších potíží zastavit na místě, nebo zabočit o více než 90°, takže pohyb bude za každých okolností uvěřitelný. Pohybuje-li se osoba rychleji, je možné její kinematiku reprezentovat jiným způsobem a zamezit tak nepřirozeným pohybům. Nejlepší možností zjednodušení je omezení úhlu zabočení objektu v závislosti na rychlosti.

Tam, kde není kladen hlavní důraz na přirozenost pohybu, ale na rychlost výpočtu, je v rámci úspory výpočetního výkonu možné zanedbat kinematiku a dynamiku i při plánování pohybu složitějších objektů. U řady z nich lze přirozenost pohybu ovlivnit polohou referenčního bodu a částečně tak zamaskovat nedokonalosti vzniklé plánováním cesty. Referenční bod v pohledu plánovacího algoritmu aproximuje naváděný objekt. Například při plánování pohybu vozidla lze referenční bod umístit na jednu z náprav, nebo do středu vozidla (v pohledu shora). V této aplikaci je nutné nalezení cesty realizovat podobně jako u běžícího člověka, tedy omezit úhel zatočení v závislosti na rychlosti. Důvodem je zachování alespoň základních principů pohybu objektu.

3.2.2 Neholonomní plánování

Na rozdíl od holonomní je neholonomní trajektorie definována tak, že závislosti mezi jednotlivými body jsou popsány soustavou nelineárních diferenciálních rovnic. Poloha jednotlivých bodů je tedy funkčně závislá na jiných veličinách, jako například rychlost pohybu objektu, nebo úhel natočení kol. Takto lze detailně popsat trajektorii objektů, u

nichž je uvažována složitější kinematika. Příkladem může být popis pohybu automobilu v kapitole o využití genetických algoritmů. Detailnější popis této problematiky je uveden v článku [6], nebo v publikaci [14]. Takovýto druh plánování je náročnější, protože je nutné ověřovat zda nalezená trajektorie vyhovuje kinematickým omezením pohybu objektu.

Jak již bylo naznačeno výše, do kategorie neholonomního plánování spadá například pohyb automobilu. Dynamika nehraje přílišnou roli, takže je možné ji zanedbat. Plánovací algoritmus spojuje jednotlivé body trajektorie pomocí křivek, které odpovídají kinematickým možnostem vozidla. Výsledná trajektorie mnohem přesněji odpovídá skutečnosti, než v případě neholonomního plánování. Pro tento typ plánování je možné využít například algoritmus RRT ve spojení s genetickými algoritmy.

3.2.3 Kinodynamické plánování

Kinodynamické plánování je nejkomplikovanější úlohou ze tří jmenovaných. Popis trajektorie je podobný jako u neholonomního plánování, v této aplikaci se však bere v úvahu i dynamika objektu. Při kinodynamickém přístupu k plánování je tedy trajektorie závislá na poloze bodů, rychlosti i hmotnosti objektu. Problém se tím stává mnohem náročnější. Kinodynamicky je nutné plánovat například trajektorii letu proudového letadla, nebo pohyb ruky manipulátoru, kde se silně projeví i dynamická omezení. Tato problematika je ale značně rozsáhlá a komplikovaná, a přesahuje rámec této práce.

3.3 Virtuální lidé

Modelování virtuálního člověka je komplexní problematikou, jíž se podrobněji zabývají autoři C. Brom a kolektiv v knize [17]. Jak je ve zmíněné publikaci uvedeno, pro vytvoření virtuální osoby je nutné uplatnit znalosti z různých oborů, jakými jsou převážně softwarové inženýrství, počítačová grafika, umělá inteligence, ale i sociologie, psychologie, nebo lingvistika.

Jednou z důležitých schopností virtuální osoby, která je předmětem této práce, je pohyb v prostoru. Kvůli uvěřitelnosti simulace se objekt musí pohybovat co nejpřirozeněji. Příkladem, který je uveden v knize Mařík a kolektiv [17], je virtuální houbař. V případě, že objeví houbu, nedojde k přeplánování jeho cesty, ale houbař pouze dojde k nalezené houbě, sebere ji a pokračuje po původní cestě. Stejně tak pokud objeví překážku, například pařez, obejde ji bez toho, aby změnil svůj směr ke globálnímu cíli jeho cesty. Takového chování je možné dosáhnout rozložením problému do dvou úrovní: plánování cesty a reaktivní navigace.

Zatímco plánování cesty slouží k naplánování globálně optimální cesty k cíli, reaktivní navigace je nízko-úrovňovou technikou, která je založena na principu potenciálových polí. Některými pravidly reaktivní navigace se podrobněji zabývá C. Reynolds v článku [21].

Pomocí této reaktivně–navigační techniky je také možné koordinovat pohyb objektů ve skupině, dosáhnout toho, aby virtuální člověk sledoval cestu nebo zeď, nebo aby postupně zpomaloval před cílem.

V prostředí s minimem překážek je možné objekt navádět pouze reaktivně, pak se objekt chová podle pravidel popsaných v kapitole 2.1. Pokud se ale jedná o rozsáhlejší prostředí, nebo o prostředí s velkým počtem překážek, pak může být reaktivní plánování neefektivní.

4 Aplikace plánovacích algoritmů

Díky rostoucímu výpočetnímu výkonu a operační paměti počítačových sestav je možné stále detailněji simulovat reálné prostředí. S tím, jak se virtuální realita přibližuje ke skutečnosti, ovšem rostou i softwarové nároky. Komplexní simulace je proto vhodné rozložit na menší podproblémy, které mohou být řešeny odděleně. Hlavní výhodou takového přístupu je efektivnější využití výkonu počítače, zvláště v případě vícejádrových platforem.

Možnosti rozložit simulaci na více oddělených částí se hojně využívá v multiagentních simulacích. Agent je softwarová jednotka, která v počítačové simulaci reprezentuje objekt (člověka, vozidlo, atd.). Jak sám název napovídá, v prostředí se mohou pohybovat stovky až tisíce agentů, kteří plní daný úkol. Účelem multiagentních simulací je výzkum komplikovaných systémů v nejrůznějších oborech, proto je kladen větší důraz na detailní popis chování agentů, nikoli na věrohodnost vzhledu prostředí a vizualizace simulace.

Opakem multiagentních simulací jsou počítačové hry, kde je kladen hlavní důraz na detailní vzhled a uvěřitelné chování virtuálního světa. Počítačové hry ve kterých je nutné plánovat pohyb osob lze dále rozdělit do dvou skupin. První skupinou jsou 3D FPS[1] hry. Počet objektů, jejichž pohyb je třeba plánovat je obvykle v řádech desítek. Druhou skupinu tvoří strategické hry, kde je počet herních postav vyšší, ale je možné je seskupovat a plánovat tak pohyb celých skupin objektů.

Ve všech jmenovaných případech je nutné řešit plánování pohybu většího počtu objektů v reálném čase. Na plánovací algoritmus jsou tedy kladeny vysoké nároky, musí být rychlý, efektivní, s co nejmenší paměťovou náročností. Efektivitu plánování do značné míry ovlivňuje nejen volba prohledávacího algoritmu, ale také volba metody diskretizace prostředí. Proto je správná funkce plánovacího algoritmu závislá na volbě vhodných přístupů k oběma problémům.

Z hlediska úspory paměti je vhodné definovat společný stavový prostor pro plánování pohybu všech objektů. V mnohých aplikacích je ale nutné plánování pohybu objektů s různými vlastnostmi, tedy nejen osob, ale například i vozidel. Společný stavový prostor by v takovém případě mohl způsobovat neefektivní plánování pohybu určitých objektů. Proto je v těchto aplikacích vhodné objekty rozdělit do skupin s podobnými vlastnostmi a vytvořit pro každou skupinu vlastní stavový prostor. Vlastnosti pro rozdělení objektů do skupin závisí na konkrétní aplikaci, obecně jimi mohou být například kinematická omezení, nebo velikost objektů. Toto doporučení se týká především metody diskretizace pomocí navigačních mřížek.

[1] FPS pochází z anglického výrazu First-person shooter. Jedná se o akční hry, ve kterých je hráči zobrazován virtuální svět z pohledu první osoby.

4.1 Plánování pohybu v multiagentních simulacích

Jak již bylo řečeno v úvodu této kapitoly, multiagentní simulace jsou primárně určeny k výzkumu chování a vzájemné interakce nejrůznějších systémů. Nároky kladené na detaily a uvěřitelnost vizualizace jsou proto velice nízké.

Z hlediska navigace osob je možné multiagentní simulace rozdělit do dvou skupin. Do první skupiny patří simulace v nekomplikovaném prostředí s malým počtem objektů, kde je možné navigaci agentů provádět pouze reaktivně, takže plánování cesty není nutné. Příkladem z první skupiny může být multiagentní simulace zaměřená na výzkum chování jedinců v menší skupině osob.

Druhou skupinou jsou multiagentní simulace v rozsáhlém, nebo komplikovaném prostředí. Vedle reaktivní navigace je v tomto případě nutné použít i plánování cesty, protože pomocí reaktivní navigace není možné zaručit dosažení globálního cíle. Konkrétním příkladem může být simulace evakuace budovy v případě požáru, kdy se lidé musí co nejrychleji dostat do bezpečí.

Diskretizace prostoru pro účely plánování pohybu v multiagentních simulacích by měla být prováděna s důrazem na vytvoření nejmenšího možného počtu stavů. Důvodem této snahy je časová a paměťová náročnost prohledávacího algoritmu. Lze předpokládat, že na malém stavovém prostoru budou paměťové i časové nároky při plánování cest minimální.

4.1.1 Projekt SMART Crowd Flow Solutions

Společnost *Buro Happold* se zabývá návrhem budov a městských prostředí. Pro vyšší efektivitu návrhu využívá multiagentní simulace *SMART Crowd Flow Solutions*. Firmou je nejprve navržen projekt budovy, prostředí je poté počítačově vymodelováno a pomocí multiagentního simulačního systému může být ověřena snadná dostupnost všech potřebných míst, požární bezpečnost a další vlastnosti. Na základě simulace může být budova upravena a je tak dosaženo maximálního pohodlí a bezpečnosti osob, které ji budou využívat. Příklad simulace je znázorněn na obrázku 4.1.

4.2 Plánování pohybu v počítačových hrách

Cílem simulací v počítačových hrách je vytvořit herní prostředí, které bude působit realisticky a současně bude mít co nejmenší možné nároky na paměť a výpočetní výkon počítače. Toho je možné dosáhnout takzvanými *level of detail* simulacemi, jejichž principem je simulace části prostředí, která je právě viditelná, nebo jinak důležitá pro správný vjem virtuálního prostředí. Ostatní části prostředí jsou simulovány pouze zjednodušeně. Tímto typem simulací se podrobněji zabývají autoři C. Brom a kolektiv v práci [4].

Obrázek 4.1: Aplikace multiagentních simulací. Obrázek byl převzat z [2]

Velká pozornost byla efektivitě plánování pohybu v počítačových hrách věnována hlavně v minulosti, kdy plánovací algoritmy využívaly nezanedbatelnou část výkonu a operační paměti tehdejších osobních počítačů. Například v populární hře *Baldur's gate* bylo možné nastavovat úroveň detailů plánování pohybu postav za účelem snížení nároků na počítačovou sestavu. V současné době je nutné zásahem do kvality plánování řešit problematiku paměťové a výpočetní efektivity prohledávacích algoritmů spíše v případech rozsáhlých simulací, které jsou popsány v předchozí kapitole. Důvodem je, že v prostředí počítačových her se vyskytuje nižší počet objektů, nebo je možné objekty slučovat do skupin. Nároky na plánování pohybu jsou proto zanedbatelné v porovnání s celkovou režií hry.

4.2.1 Akční hry

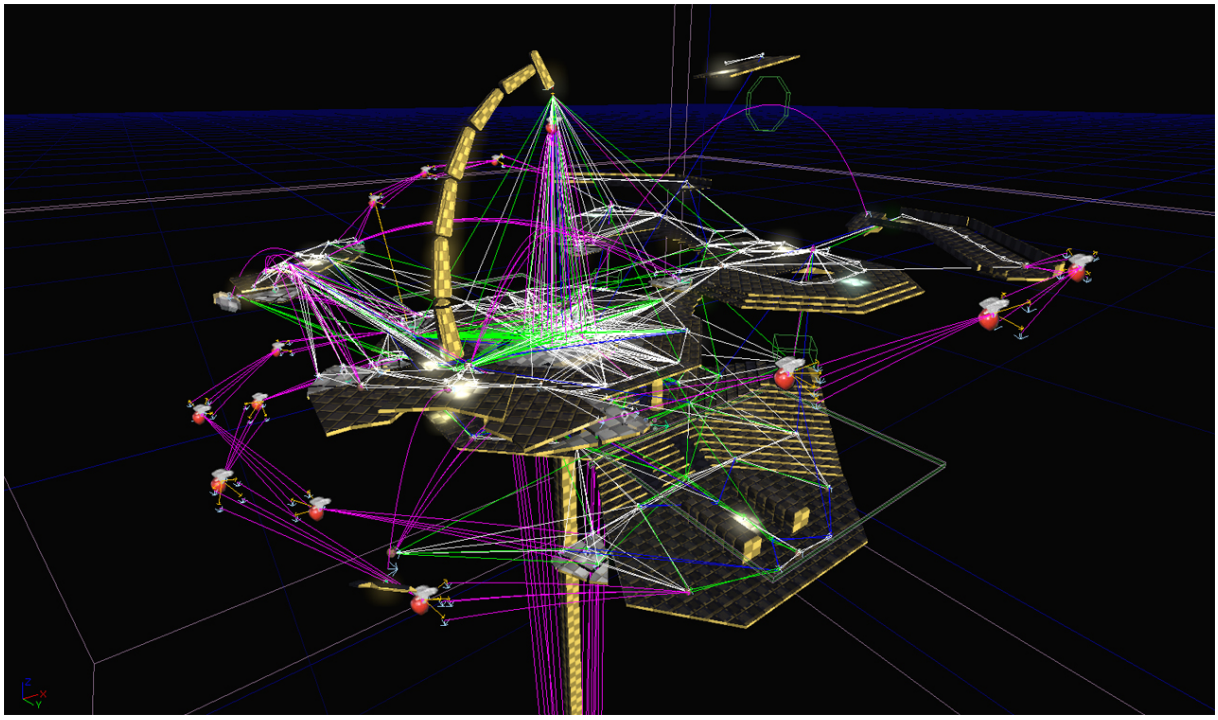
Tento typ her je velice oblíbený, protože hráče začleňují do prostředí a děje hry. Hráči je virtuální svět zobrazován zpravidla z pohledu první osoby. To znamená, že virtuální prostředí je na obrazovce zobrazováno tak, jak by jej vnímal člověk, který se v něm skutečně pohybuje. Tento přístup přináší výhody i při simulaci samotného prostředí, kdy lze s výhodou použít přístup *level of detail*, který je zmíněn v úvodu této podkapitoly.

Počet protivníků v FPS hrách výjimečně může dosáhnout několika desítek. Virtuální prostor ve kterém se mohou postavy pohybovat obvykle bývá malý a omezený překážkami, jako je tomu například ve hře *Half life 2*. Stále rostoucí výkon osobní výpočetní techniky umožnil chápat protivníky v počítačových hrách jako virtuální osoby. Agenti, kteří simulují protivníky jsou tedy schopni reagovat na hráče, vytvářet vlastní taktiku a realizovat se v týmu. Ukázkou takového chování virtuálních protivníků je akční hra *Unreal tournament 4 (UT4)*.

Plánování pohybu je v tomto typu her realizováno přesně tak, jak je popsáno v kapitole 3.3. Reaktivní navigace je použita pro lokální účely a splnění cíle. Příkladem z

výše zmíněné hry *UT4* může být zneškodnění soupeře, pak reaktivní pravidla přitahují agenta k soupeřům. Zatímco jiný agent má úkol získat vlajku, takže se naopak ostatním agentům snaží vyhnout a dostat se co nejrychleji a nejbezpečněji k vlajce. V takovémto případě je nutné provést i naplánování cesty.

Vytvoření diskrétního modelu herní mapy pro účely plánování cesty a následné prohledávání může být provedeno kombinacemi přístupů popisovaných v úvodu této práce. Studia, která vyvíjejí počítačové hry, svá řešení pochopitelně tají. Podle chování agentů je někdy možné odhadnout, jakým způsobem byl jejich pohyb naplánován. Asi nejběžnějšími metodami diskretizace prostředí jsou navigační mřížky, nebo navigační sítě. Volba vhodného prohledávacího algoritmu pak závisí na zvolené metodě diskretizace. Pokud bude zvolena diskretizace pomocí navigačních mřížek, pak lze s výhodou použít některý z any-angle algoritmů. V případě, že bude zvolena metoda diskretizace pomocí navigační sítě, pak lze pro prohledávání použít například algoritmus A*. Na obrázku 4.2 je znázorněn příklad diskretizace herní mapy v 3D počítačové hře *Unreal Tournament*.



Obrázek 4.2: Příklad diskretizace prostoru v akčních počítačových hrách. Obrázek byl převzat z [1]

4.2.2 Strategické hry

Virtuální svět je ve strategických hrách typicky zobrazován v pohledu shora, nebo z perspektivy. Na rozdíl od výše zmíněných akčních her je tedy zobrazována větší část

prostředí. Virtuální prostředí bývá rozlehlejší, čtvercového, nebo obdélníkového tvaru. S výhodou tedy lze pro účely plánování cesty využít metodu diskretizace prostředí pomocí čtvercové navigační mřížky, která pokryje prostředí bezezbytku. Cílem hry je vytvořit fungující město, kde se může pohybovat až několik set objektů.

Vliv většího počtu objektů, jejichž cesty je nutné plánovat je kompenzován tím, jak hráč určuje jejich pohyb. Nejprve označí skupinu objektů, která má být přemístěna a následně kliknutím do prostředí určí, na jaké místo se objekty mají přesunout. Plánování cesty je pak možné provádět pro celé skupiny. Hlavní výhodou je, že nemůže nastat situace, kdy by bylo nutné plánovat cestu velkého množství objektů ve stejný okamžik. Koordinace pohybu objektů ve skupině je realizována pomocí reaktivních pravidel. Stejně tak interakce objektů s okolním virtuálním světem je ošetřena reaktivně. Jako příklad interakce objektů s prostředím je možné považovat průchod skupiny objektů skrz město. Skupina se rozdělí do několika proudů objektů, které jsou schopné město překonat bez kolize a na druhém konci města se opět seskupí do původní formace.

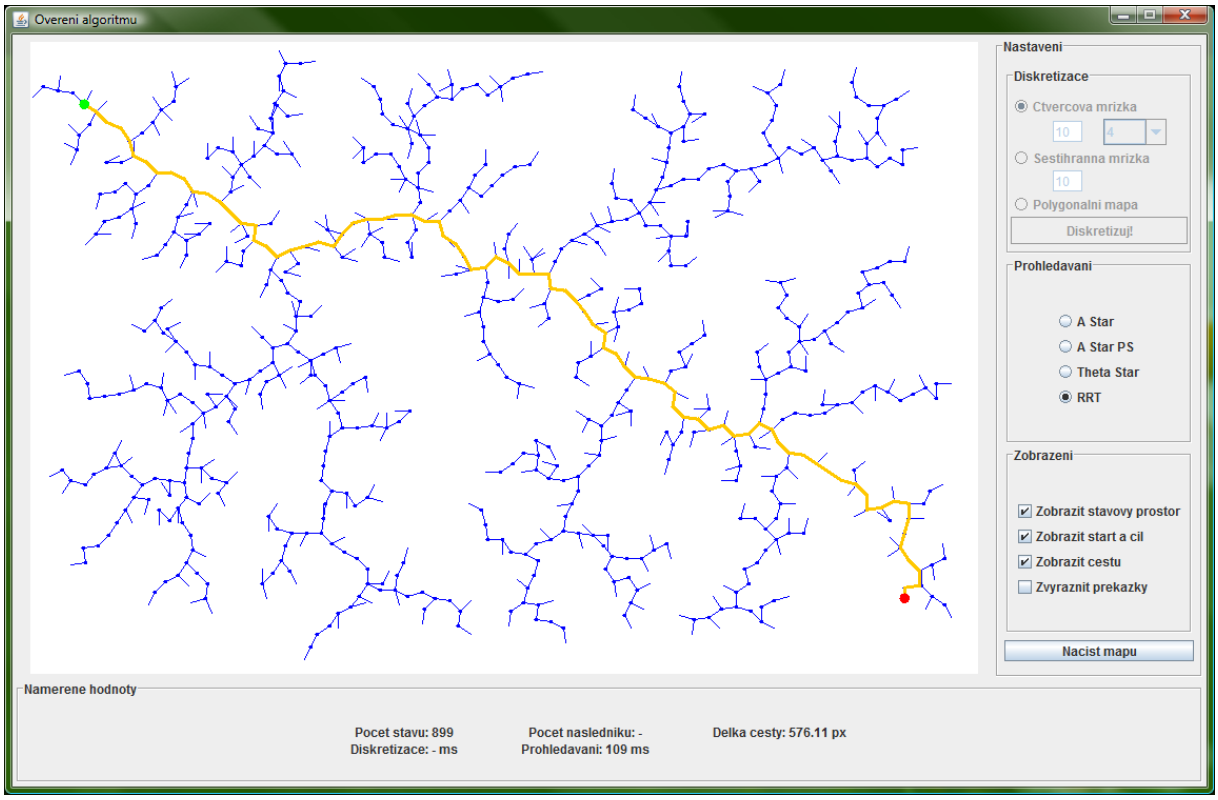
Ve většině her, jako je například *Age of Empires*, je možné stavět budovy libovolně v celém prostředí. Komplikovanějším řešením takového problému je průběžně zanášet budovy do diskretizovaného stavového prostoru a počítat s nimi již při plánování cesty. Mnohem vhodnější řešení je pomocí reaktivních pravidel agenta zaručit, aby se jakékoli budově dokázal vyhnout. Speciálním případem je opevnění, které je naopak vhodné zanést do stavového prostoru a počítat s ním již při plánování cesty, aby se objekty nepokoušely opevnění obcházet.

Efektivně je tato problematika vyřešena ve hře *The Battle for the Middle Earth*. Prostředí je navrženo tak, že jsou přesně určena místa pro výstavbu měst, v nichž jsou pevně určeny pozice budov. Předem je také dán tvar města a způsob opevnění. Terén mapy v této hře není příliš komplikovaný, v prostředí se obvykle vyskytuje několik kopců a řeka. Stavový prostor pro plánování cesty je tedy možné vytvořit pomocí navigační sítě, s malým počtem stavů. Velká část navigace může být realizována pomocí reaktivních navigačních pravidel.

5 Ověření vlastností plánovacích algoritmů

5.1 Popis aplikace

Tato kapitola je zaměřena na praktické ověření předpokládaných vlastností vybraných prohledávacích algoritmů a diskretizačních metod. Pro tyto účely byla v programovacím jazyku Java implementována aplikace, která je schopna měřit čas potřebný pro nalezení cesty zvoleným algoritmem, čas potřebný pro diskretizaci prostředí, počet vytvořených stavů, počet následníků stavu a délku nalezené cesty. Měření paměti využité pro jednotlivé úkony je prováděno pomocí aplikace *NetBeans Profiler*, která je součástí vývojového prostředí *NetBeans*.



Obrázek 5.1: Snímek okna aplikace pro ověření vlastností algoritmů

Do aplikace je při jejím spuštění načtena základní mapa prostředí ve formě bitmapy. Při běhu aplikace je možné nahrát libovolnou mapu prostředí pomocí tlačítka ,,Načíst mapu", které je umístěno na panelu ,,Nastavení"v pravé části okna programu. Mapa prostředí ve formátu PNG musí splňovat následující požadavky: překážky jsou polygony vyplněné černou barvou na bílém pozadí, hrany překážek musí být rovnoběžné s osou x, osou y, nebo jednou z diagonál. V případě, že je mapa menší než zobrazovací plocha, je zvětšena, aby lépe vyplnila zobrazovací prostor. Na pravém panelu ,,Nastavení"je také možné nalézt panel, na kterém lze pomocí přepínačů nastavit metodu diskretizace.

V případě výběru metody diskretizace pomocí navigačních mřížek lze nastavit velikost strany mřížky, u čtvercové mřížky je navíc možné vybrat počet sousedů buňky. Dalším panelem nastavení je panel pro výběr metody prohledávání. Nalezení cesty je realizováno automaticky po zadání cílového stavu. Počáteční a cílový stav je možné vybrat kliknutím do plochy mapy. Pokud jsou zadány počáteční i koncový stav a je vybrán jiný prohledávací algoritmus, cesta se automaticky přepočítá nově vybraným algoritmem. Výjimkou je algoritmus RRT, který při hledání cesty vytváří stavový prostor a proto je stávající stavový prostor i s aktuálními koncovými stavy vymazán. Po zadání nového počátečního, i cílového stavu je nalezena cesta. Při opětovném kliknutí do prostoru mapy, po nalezení cesty algoritmem RRT, je pouze u této metody spuštěno nové prohledávání. Posledním panelem nastavení je panel ,,zobrazení", kde je možné nastavit, co vše má být v mapě zvýrazněno. Ve spodní části okna programu je umístěn panel ,,Naměřené hodnoty", kde jsou zobrazovány všechny údaje, které je možné měřit pomocí této aplikace.

Jak již bylo řečeno, měření použité paměti je možné provést pomocí další aplikace, jako je například *Java Visual VM*, nebo v tomto případě *NetBeans Profiler*. Všechny informace o stavovém prostoru jsou ukládány jako *List* objektů typu *State* a podobně jsou informace o větvených stavech při prohledávání ukládány v objektech typu *AStarState*. Výjimku tvoří algoritmus RRT, který při prohledávání konstruuje stavový prostor a informace o stavech ukládá jako objekty typu *State* a hrany jsou uloženy jako množina objektů typu *Line2D*.

Pro účely měření paměťových požadavků byl software vždy nastaven tak, aby zaznamenával pouze vytvoření objektu, nikoli jeho odstranění. *Garbage collector* proto neovlivní výsledky měření případným odstraněním objektu.

Pro účely ověření vlastností byly použity mapy o velikosti 450×300 px.

5.2 Prováděné experimenty

Následující experimenty mají za cíl ověření předpokládaných vlastností jednotlivých metod plánování cesty. Tato měření mohou být využita jako podklady při výběru vhodných přístupů k plánování pohybu v nejrůznějších aplikacích.

5.2.1 Ověření vlastností diskretizace pomocí polygonálních map

5.2.1.1 Návrh experimentu

Cílem experimentu bylo ověření vlastností diskretizace pomocí navigačních map. Ověřovanými vlastnostmi byly časová a paměťová náročnost této metody v závislosti na počtu překážek v prostředí. V druhé části experimentu byla zjišťována paměťová a časová náročnost prohledávacího algoritmu, také v závislosti na počtu překážek v prostředí.

5.2.1.2 Realizace experimentu

Pro účely měření byla do programu načítána prostředí s daným počtem překážek, v tabulce 5.1 označovaným n. Každá překážka měla osm hran, jak je znázorněno na obrázku 5.2.

Obrázky s mapami prostředí a určitým počtem překážek byly předem ručně připraveny. Během načítání byl aplikací měřen čas t_p potřebný pro nalezení černých míst, která reprezentují překážky, v obrázku. Po načtení byla mapa diskretizována pomocí polygonální mapy a byl změřen čas diskretizace t_d a paměť m_d využitá pro vytvoření stavového prostoru. Poté byl zadán počáteční stav, který byl volen v pravém horním rohu prostředí, cílový stav, který byl volen v levém spodním rohu prostředí, a byla naměřena doba t a paměť m_s potřebná pro nalezení cesty algoritmem A*. Všechna měření byla provedena pětkrát a do tabulky naměřených hodnot byla vždy zanesena průměrná hodnota z naměřených údajů.

Výsledky druhé části tohoto experimentu, viz tabulka 5.2, je možné aplikovat na metodu diskretizace pomocí polygonálních map, nebo na metodu ruční diskretizace pomocí *way-point* grafu. Ve všech jmenovaných případech je vytvářen graf s podobnými vlastnostmi, popisující dané prostředí.

Obrázek 5.2: Tvar překážky

n	Počet stavů	t_p [ms]	t_d [ms]	m_d [kB]
16	128	1 784	122	3
24	192	2 342	254	4
42	336	3 464	676	7
74	592	4 906	2035	12
86	688	5 911	2467	14

Tabulka 5.1: Závislost využité paměti a času potřebného pro diskretizaci prostředí pomocí polygonálních map na počtu překážek.

5.2.1.3 Zhodnocení experimentu

Měřením bylo ověřeno, že časové i paměťové nároky metody diskretizace pomocí polygonálních map rapidně stoupají s rostoucím počtem překážek, a proto je vhodné metodu používat pouze v prostředí s malým počtem překážek.

Výhodou této metody je doba potřebná pro nalezení cesty, viz tabulka 5.2. Paměťové i časové nároky pro nalezení cesty jsou ve srovnání s metodou diskretizace pomocí na-

n	t [ms]	m_s [B]
16	< 1	128
24	$1,3$	565
42	$2,3$	1504
74	$10,0$	5963
86	$12,4$	8544

Tabulka 5.2: Ověření závislosti využité paměti a času potřebného pro nalezení cesty na počtu překážek.

vigačních mřížek minimální. Naopak nevýhodou této metody je nízká rozlišovací schopnost. Protože přechody mezi stavy jsou vedeny přes celý volný prostor, informace o ceně cesty může být značně zkreslená. Polygonální mapy neumožňují detailní plánování. Tento typ diskretizace je vhodný především pro popis rozsáhlého prostředí, kde není třeba plánovat pohyb detailně. Například plánování cesty virtuálního člověka, kde naplánovaná cesta slouží pouze pro směřování objektu k cíli.

Vzhledem k rychlému nalezení cesty jsou uvedené metody diskretizace, ve spojení s algortimem A*, vhodné pro plánování pohybu velkého počtu objektů.

5.2.2 Ověření vlastností diskretizace pomocí navigačních mřížek

5.2.2.1 Návrh experimentu

Cílem tohoto experimentu bylo ověřit a porovnat vlastnosti navigačních mřížek v závislosti na velikosti hrany buněk. Ověřovanými vlastnostmi byly především paměťová a časová náročnost diskretizace prostředí, které úzce souvisí s počtem vytvořených stavů. Dále byly navigační mřížky porovnány z hlediska efektivity pokrytí členitého prostředí.

5.2.2.2 Realizace experimentu

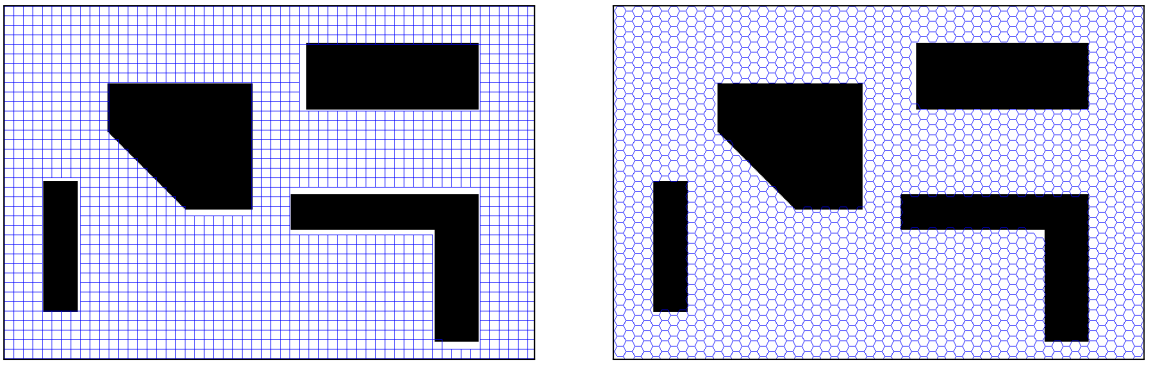
Pro účely měření bylo použito prostředí bez překážek, protože k popisu volného prostředí je třeba největší možný počet stavů a na diskretizaci pomocí navigačních mřížek jsou proto kladeny nejvyšší nároky. Prostředí bylo postupně popisováno čtvercovou mřížkou s uvažováním čtyř sousedních stavů, čtvercovou mřížkou s uvažováním osmi sousedních stavů a šestihrannou mřížkou s velikostí hrany buněk a. Při tomto experimentu byla měřena závislost doby potřebné k diskretizaci prostředí příslušnou mřížkou, která je v tabulce 5.3 označena jako t, a dále paměti m použité při diskretizaci, na velikosti hrany buněk. Každé měření použité paměti i potřebného času bylo provedeno pětkrát pro jeden rozměr mřížky. Do tabulky 5.3 byla vždy zanesena průměrná hodnota z naměřených hodnot.

Pro porovnávání efektivity pokrytí členitého prostoru navigačními mřížkami byla do programu načtena mapa prostředí s překážkami. Prostředí bylo popsáno pomocí

čtvercové a šestihranné mřížky tak, aby diskretizací oběma metodami vznikl obdobně veliký stavový prostor. Velikost hrany buněk u obou typů mřížky byla určena podle tabulky 5.3 na $a_{hex} = 5$ px, pro šestihrannou mřížku, a $a_{ctv} = 8$ px, pro čtvercovou mřížku. Poté byla v obou případech provedena diskretizace a následně byl sejmut snímek diskretizovaného prostředí.

a	Čtvercová mřížka					Šestihranná mřížka		
	Počet stavů	čtyři sousedé		osm sousedů				
		t [ms]	m [kB]	t [ms]	m [kB]	Počet stavů	t [ms]	m [kB]
2	33 750	52 081	1 109	94 838	1 990	12 814	15 232	798
3	15 000	10 202	492	19 610	882	5 693	3 638	354
4	8 400	3 310	275	6 148	658	3 225	1 410	200
5	5 400	1 463	177	2 732	317	2 006	449	124
6	3 750	726	123	1 259	220	1 397	250	86
7	2 688	395	88	693	157	1 029	155	63
8	2 072	265	67	429	121	777	107	48
9	1 650	162	53	294	96	627	76	39
10	1 350	122	44	203	79	493	72	30

Tabulka 5.3: Ověření vlastností diskretizace prostředí pomocí navigačních mřížek.



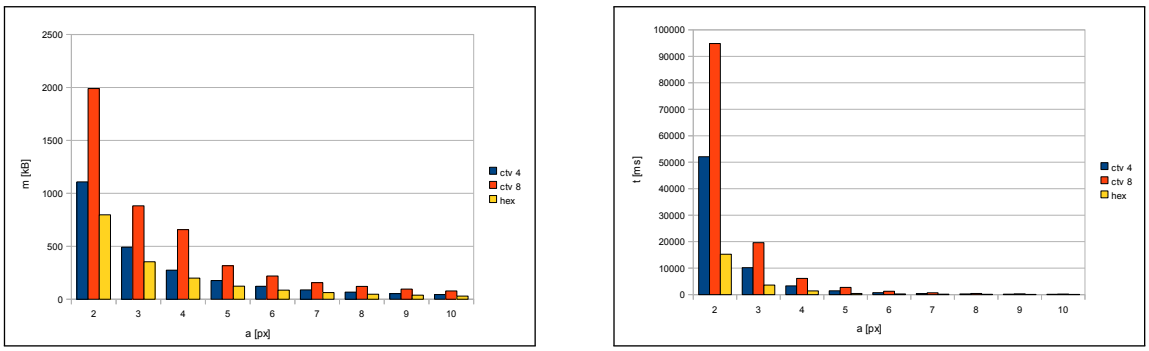
(a) Čtvercová mřížka s hranou velikosti 8px (1550 stavů)

(b) Šestihranná mřížka s hranou velikosti 5px (1506 stavů)

Obrázek 5.3: Porovnání popisu prostředí pomocí navigačních mřížek

5.2.2.3 Zhodnocení experimentu

Jak je patrné z grafů 5.4, časové i paměťové nároky na popis prostředí pomocí navigačních mřížek rapidně klesají s rostoucí velikostí hrany mřížky. Další, neméně důležitou vlastností je schopnost mřížky efektivně pokrýt členitý prostor, aby plánování pohybu mohlo probíhat detailně. Hlavní výhodou této metody popisu prostředí je, že s jednotlivými buňkami mřížky je možné snadno spojit nejrůznější informace a plánovat tak cesty podle více kritérií.



(a) Využití paměti (b) Potřebný čas

Obrázek 5.4: Grafické znázornění naměřených hodnot

Na obrázku 5.3(b) je znázorněna diskretizace prostředí pomocí šestihranné mřížky, vzniklý stavový prostor obsahuje 1 506 stavů. Na obrázku 5.3(a) je znázorněna diskretizace stejného prostředí pomocí čtvercové mřížky, stavový prostor vzniklý touto metodou obsahuje 1 550 stavů. Z porovnání obou obrázků plyne, že pomocí šestihranné mřížky lze efektivněji a lépe popsat členitý prostor, protože nepopsaná volná plocha v okolí překážek je ve srovnání s čtvercovou mřížkou menší.

Pomocí tohoto experimentu bylo potvrzeno, že pomocí šestihranné mřížky lze efektivně a rychleji vyplnit členitý prostor s pomocí menšího počtu stavů, než u čtvercové mřížky. Naopak čtvercová mřížka s vhodně zvolenou velikostí hrany buňky je vhodnější pro popis prostoru s osově zarovnanými pravidelnými překážkami.

5.2.3 Porovnání algoritmů A* a Theta*

5.2.3.1 Návrh experimentu

Cílem tohoto experimentu bylo porovnání vlastností prohledávacích algoritmů Theta*, A* a A* s vyhlazováním. Při tomto experimentu byla měřena časová a paměťová náročnost algoritmů v závislosti na typu mřížky a velikosti hrany buněk. Dalším porovnávaným údajem byla délka nalezené cesty. Tento experiment byl rozdělen na tři části. Cílem první části bylo porovnání paměťových a časových nároků uvedených algoritmů v případě, že počáteční a cílový stav jsou navzájem viditelné. Cílem druhé a třetí části experimentu bylo ověření vlastností algoritmů na mapě s překážkami.

5.2.3.2 Realizace experimentu

Jednotlivým částem experimentu odpovídají bloky v tabulkách naměřených hodnot. Všechny hodnoty byly měřeny pětkrát a do tabulky byla vždy zanesena průměrná hodnota z hodnot získaných měřením. V závislosti na velikosti a typu navigační mřížky byly měřeny následující údaje: čas potřebný k nalezení cesty t, paměť využitá při prohledávání

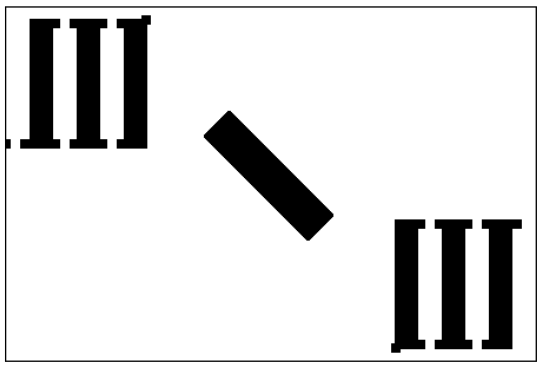
m a délka nalezené cesty l. u algoritmu A* s vyhlazováním byla navíc měřena doba potřebná pro vyhlazení cesty t_s. Jako heuristická funkce je u všech algoritmů použita Euklidovská vzdálenost právě expandovaného stavu a cílového stavu.

První část experimentu, uvedená v bloku tabulek naměřených hodnot označeném ,,Experiment č.1", byla měřena na mapě bez překážek. Celý prostor byl diskretizován navigační mřížkou, která je blíže určena v popisu příslušné tabulky, s velikostí hrany buněk a. Poté byly pro každý řádek v tabulce zadávány stejné koncové stavy cesty. Počáteční stav byl vždy volen v pravém horním rohu mapy a koncový stav v levém spodním rohu mapy.

Pro naměření dat uvedených ve zbývajících blocích, tedy ,,Experiment č.2"a ,,Experiment č.3", bylo do aplikace načteno prostředí, které je znázorněno na obrázku 5.5. Pro jednotlivé řádky v tabulkách byla měření opět spouštěna s totožnými koncovými body cest v prostředí diskretizovaném pomocí navigační mřížky s velikostí hrany a.

Počáteční stav cesty byl v druhé části experimentu volen v pravém horním rohu, koncový stav byl volen v levém spodním rohu. Cestu mezi těmito stavy je možné v daném prostředí snadno nalézt, protože heuristická funkce využívaná oběma algoritmy se blíží skutečné vzdálenosti. Z tohoto důvodu je možné předpokládat, že bude expandován menší počet stavů, než ve třetí části experimentu.

Počáteční stav cesty byl ve třetí části experimentu volen v levém horním rohu a koncový stav byl volen v pravém spodním rohu. Tyto dva stavy od sebe dělí velký počet překážek, které způsobují značnou chybu odhadu heuristické funkce algoritmů. Stavový prostor, který je nutné prohledat pro nalezení nejkratší cesty je proto rozsáhlejší, než je tomu v předchozím případě.



Obrázek 5.5: Mapa prostředí

5.2.3.3 Zhodnocení experimentu

Experimenty úspěšně potvrdily předpokládané vlastnosti algoritmů. Algoritmus Theta* až na dvě výjimky pokaždé nalezl kratší nebo stejně dlouhou cestu, v porovnání s al-

a	A*			A* PS				Theta*		
	t [ms]	m [kB]	l [px]	t [ms]	t_s [μs]	m [kB]	l [px]	t [ms]	m [kB]	l [px]
Experiment č.1										
2	34 919	313	706, 0	34 627	21 450	314	510, 4	823	48	510, 4
6	579	34	690, 0	581	5 871	34	499, 3	96	9	499, 3
10	109	12	690, 0	77	3 538	12	499, 3	42	4	499, 3
Experiment č.2										
2	27 220	278	738, 0	27 945	11 601	279	552, 7	3 328	168	545, 2
6	393	28	738, 0	375	4 457	28	554, 6	169	22	545, 5
10	54	10	730, 0	56	2 607	10	548, 6	55	8	541, 6
Experiment č.3										
2	28 067	278	738, 0	27 923	14 646	279	599, 2	20 849	476	598, 5
6	359	28	738, 0	367	4 745	28	601, 9	561	50	601, 5
10	42	8	730, 0	41	4 166	8	602, 4	117	13	599, 9

Tabulka 5.4: Porovnání algoritmů ve spojení s metodou diskretizace pomocí čtvercové mřížky s uvažováním čtyř sousedů stavu.

a	A*			A* PS				Theta*		
	t [ms]	m [kB]	l [px]	t [ms]	t_s [μs]	m [kB]	l [px]	t [ms]	m [kB]	l [px]
Experiment č.1										
2	19 158	327	565, 8	18 830	15 099	327	532, 5	544	43	532, 5
6	262	35	548, 8	257	4 891	35	515, 9	78	8	515, 9
10	41	13	560, 1	40	2 615	13	527, 0	38	4	526, 9
Experiment č.2										
2	13 919	292	554, 3	13 797	8 230	292	529, 9	5 388	316	529, 0
6	236	33	572, 8	232	2 638	33	546, 2	273	38	545, 5
10	34	12	565, 9	34	1 547	12	540, 3	99	14	541, 6
Experiment č.3										
2	31 275	432	616, 2	31 238	5 064	432	600, 6	38 728	952	597, 5
6	412	43	618, 5	413	1 620	43	607, 3	994	95	599, 1
10	26	9	601, 1	26	939	9	590, 4	163	23	589, 4

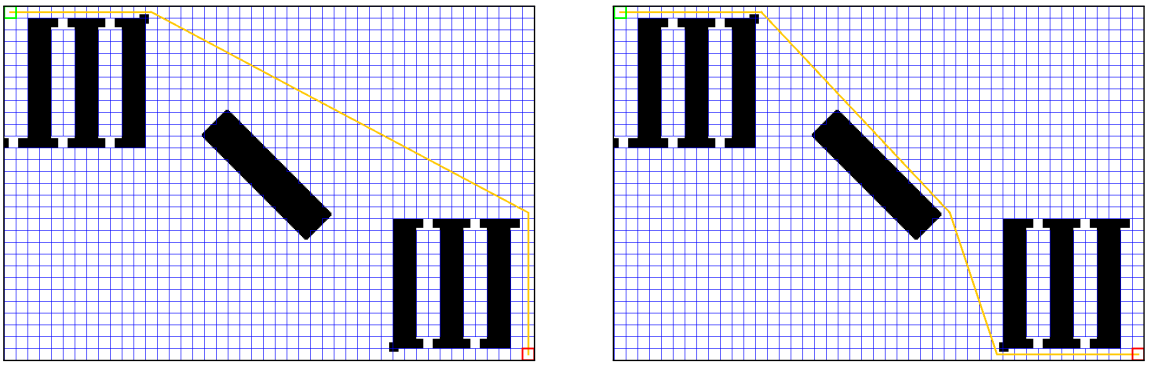
Tabulka 5.5: Porovnání algoritmů ve spojení s metodou diskretizace pomocí čtvercové mřížky s uvažováním osmi sousedů stavu.

goritmem A* s vyhlazováním. Chyba algoritmu Theta* je způsobena algoritmem jako takovým, bližší informace je možné nalézt v popisu algoritmu v [18].

V závislosti na hodnotách uvedených v tabulkách naměřených hodnot je možné říci, že pokud heuristická funkce dobře popisuje skutečnost, algoritmus Theta* při prohledávání expanduje výrazně méně stavů, než algoritmus A*. Proto v tomto případě nalezne cestu v kratším čase a s výrazně menšími paměťovými nároky. Jak dokládají porovnání naměřených hodnot při realizaci experimentů č.1 a č.2, v tabulkách naměřených hodnot.

a	A*			A* PS				Theta*		
	t [ms]	m [kB]	l [px]	t [ms]	t_s [μs]	m [kB]	l [px]	t [ms]	m [kB]	l [px]
Experiment č.1										
2	655	44	547,3	624	10 753	44	529,3	141	11	521,3
6	14	5	540,4	13	3 294	5	520,9	23	2	520,9
10	3	2	519,6	3	2 101	2	503,2	11	1	503,2
Experiment č.2										
2	2 828	101	599,3	2 791	5 695	101	543,8	1 027	95	543,3
6	40	41	592,4	41	2 090	11	537,2	80	11	536,0
10	9	4	588,9	9	1 239	4	526,0	33	3	524,4
Experiment č.3										
2	3 342	116	623,5	3 383	7 006	115	598,7	4 934	266	595,3
4	61	13	616,6	62	2 889	13	592,3	292	30	593,2

Tabulka 5.6: Porovnání algoritmů ve spojení s metodou diskretizace pomocí šestihranné mřížky.



(a) Cesta nalezená algoritmem A* PS (b) Cesta nalezená algoritmem Theta*

Obrázek 5.6: Porovnání cest nalezených ve čtvercové mřížce pomocí algoritmů A* s vyhlazováním a Theta*

Pokud je skutečná vzdálenost stavů vyšší, než vzdálenost odhadovaná pomocí heuristické funkce, oba algoritmy pro nalezení cesty expandují velký počet stavů. Tímto je způsoben nárůst paměti i času potřebného k nalezení cesty oběma algoritmy při realizaci experimentu č.3. Algoritmus Theta* v tomto případě nalezne cestu pomaleji, protože při expanzi stavů navíc vykonává testy viditelnosti. Stále ale nalézá v porovnání s algoritmem A* s vyhlazováním kratší cesty.

Jak ilustruje porovnání nalezených cest na obrázku 5.6, největší rozdíl mezi optimální cestou která je vázána k mřížce a cestou nalezenou pomocí algoritmu Theta*, je možné pozorovat na čtvercové mřížce s uvažováním čtyř sousedních stavů. Dalším z omezení této metody diskretizace je, že při jejím použití pro popis prázdného prostředí je paměťová a časová náročnost algoritmu A* vyšší, než při opakování stejného experimentu v prostředí s překážkami, viz tabulka 5.4, experiment č.1 a experiment č.2. Tato skutečnost je

způsobena tím, že změna směru cesty je mřížkou omezena na násobky pravého úhlu. Při diagonálním směru prohledávání je proto nutné expandovat nadměrný počet stavů.

Třetí experiment při použití šestihranné mřížky nebylo možné realizovat se stejným nastavením velikosti hrany mřížky, jaké bylo použito u ostatních experimentů. Důvodem byla skutečnost, že překážky byly osově zarovnané a stavy s hranou větší než $a = 4$ px nebyly propojeny až do rohů použité mapy.

5.2.4 Ověření vlastností algoritmu RRT

5.2.4.1 Návrh experimentu

Tento experiment byl realizován za účelem ověření vlastností algoritmu RRT.

5.2.4.2 Realizace experimentu

Pro účely experimentu byl do aplikace přidán cyklus s pevným počtem 100 kroků, aby naměřená data měla dostatečnou vypovídající hodnotu. Cyklus byl automaticky spuštěn při požadavku přeplánování cesty. V každé iteraci proběhlo přeplánování cesty pomocí algoritmu RRT a na standardní výstup byly vypisovány následující údaje: čas potřebný k nalezení cesty t, délka nalezené cesty l a počet vytvořených stavů n. Naměřená data byla následně přenesena do programu *OpenOffice Calc*, kde byla statisticky zpracována.

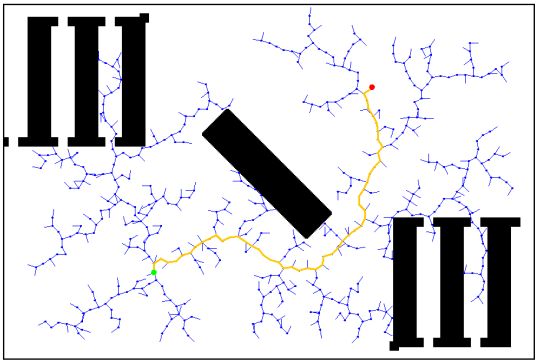
Do aplikace bylo nejprve načteno prostředí bez překážek a náhodně byly zadány počáteční a koncový stav. Vzdálenost zadaných stavů, naměřená aplikací byla $l = 240,6$ px. Po naměření 100 hodnot bylo do aplikace načteno stejné prostředí, které bylo využito pro účely ověření vlastností algoritmů v předchozí podkapitole a celý proces byl opakován. Zvolené koncové stavy, spolu s příkladem nalezené cesty jsou znázorněny na obrázku 5.7. Obrázek byl pořízen sejmutím obrazovky po proběhnutí experimentu. Výsledky zpracování naměřených dat jsou uvedeny v tabulce 5.7.

	Volné prostředí			Prostředí s překážkami		
	t [ms]	l [px]	n	t [ms]	l [px]	n
min	2	274,1	94	5	335,9	169
max	8 075	470,4	6 729	6 333	560,9	5 722
med	205	322,1	1 287	150	389,3	1 098
avg	605,6	332,5	1 585	437,3	399,0	1 366

Tabulka 5.7: Ověření vlastností algoritmu RRT.

5.2.4.3 Zhodnocení experimentu

Na obrázku 5.7 je možné pozorovat rapidní expanzi prohledávacího stromu algoritmu RRT do volného prostoru. Z naměřených dat, uvedených v tabulce 5.7 vyplývá, že algo-



Obrázek 5.7: Příklad cesty nalezené algoritmem RRT

ritmus RRT ve volném prostoru nalezl cestu maximálně dvakrát delší, než je optimální cesta. Doba prohledávání se v obou případech pohybuje maximálně v řádech jednotek sekund. Mediány časů prohledávání jsou v porovnání s maximy řádově nižší, z toho je možné usoudit, že pokud by algoritmus RRT ,,hlídal"dobu strávenou prohledáváním a po dosažení určité prahové hodnoty by bylo prohledávání restartováno, maximální časy prohledávání by byly významně nižší.

Z naměřených hodnot dále vyplývá, že plánování cesty proběhlo rychleji v prostředí s překážkami. Tato skutečnost je způsobena především tím, že expanze prohledávacího stromu algoritmu do stran byla překážkami omezena.

Tento algoritmus by mohl najít uplatnění při plánování pohybu v multiagentních simulacích, kde je třeba plánovat cesty náhodně. Příkladem může být multiagentní simulace lidí prchajících z hořícího stadionu.

6 Závěr

Tato práce se zabývá plánováním pohybu ve virtuálním 2D prostředí. První část je zaměřena na teorii plánování pohybu ve virtuálním prostředí. Problematika plánování je velice rozsáhlá a proto je tato kapitola rozdělena z hlediska vlastností objektů, jejichž pohyb je plánován, na plánování cest a plánování trajektorií. Popis plánování cest je dále rozdělen na plánování ve spojitém prostředí, kde je popsán jeden z hlavních zástupců této kategorie, algoritmus navigace pomocí potenciálových polí. Pro plánování cesty v diskrétním prostředí je nutné spojité prostředí nejprve vhodně diskretizovat, proto jsou dále popsány běžně používané metody diskretizace prostředí. Stavový prostor vzniklý diskretizací je možné prohledávat a tak v něm plánovat cestu pomocí některého z popsaných plánovacích algoritmů.

Plánování trajektorií je komplexnější úlohou, která je využitelná spíše při plánování pohybu objektů s kinematickými nebo dynamickými omezeními. Proto je v této práci uveden pouze jeden přístup k řešení problematiky plánování trajektorie. Dále jsou v první části práce popsána omezení plánovacích algoritmů, jako je například proměnlivé prostředí, nebo 2,5D prostředí.

Další částí této práce je popis aplikací plánovacích algoritmů v multiagentních simulacích a počítačových hrách. Klíčovým rozdílem mezi těmito dvěma kategoriemi je účel simulace a počet simulovaných objektů. Předmětem multiagentní simulace je výzkum určitého systému, kde na uvěřitelnost vzhledu simulace není kladen důraz. V multiagentních simulacích proto není kladen důraz na detailní plánování cesty, ale spíše na rychlost plánování. Oproti tomu v počítačových hrách má nejvyšší prioritu uvěřitelnost vzhledu simulace. Plánování pohybu v počítačových hrách je tedy prováděno s důrazem na přirozenost pohybu. Poslední část práce je věnována experimentům, které byly realizovány za účelem ověření předpokládaných vlastností plánovacích technik. Prvním prováděným experimentem bylo ověření vlastností metody diskretizace pomocí polygonálních map. Měřením byly ověřeny paměťové a časové nároky této metody diskretizace v závislosti na počtu překážek v prostředí. Dále bylo ověřeno, že doba nalezení cesty ve vzniklém grafu se dramaticky nemění. V druhém experimentu byly porovnány vlastnosti jednotlivých typů navigačních mřížek, z ověření plyne, že časové a paměťové nároky diskretizace prostředí rapidně rostou se zmenšující se velikostí hrany buněk. Na základě porovnání optimality popisu prostoru je možné říci, že šestihranné mřížky lépe popisují obecně uspořádané překážky. Pokud jsou ale překážky popsány pomocí čtyřúhelníků zarovnaných podle os souřadnicového systému, je vhodněješí použít čtvercové mřížky. Třetí experiment byl realizován za účelem porovnání plánovacích algoritmů A*, A* s vyhlazováním a Theta*. Na základě experimentu bylo ověřeno, že algoritmus Theta* nalezne přirozeněji vypadající cestu, která je stejně dlouhá, nebo kratší v porovnání s cestou

nalezenou pomocí algoritmu A* s vyhlazováním. Poslední experiment měl za cíl ověřit vlastnosti algoritmu RRT. Při použití ve volném prostředí byla cesta nalezená pomocí algoritmu RRT maximálně dvakrát delší v porovnání se skutečnou vzdáleností bodů. Doba pro nalezení cesty se pohybuje maximálně v řádech jednotek sekund, medián doby prohledávání je však o řád nižší. Všechna ověření dopadla dle teoretických očekávání. Na základě výsledků realizovaných experimentů může být vhodně zvolena metoda plánování pohybu v nejrůznějších aplikacích.

7 Literatura

[1] http://dantennant.com/images/Tutorial/BotPaths.jpg [online].

[2] Smart solution networks. http://www.smart-solutions-network.com/page/smart-move-viz-introduction [online].

[3] D. Bertsimas a J. Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1):10–15, 1993.

[4] C. Brom, O. Šerý, a T. Poch. Simulation level of detail for virtual humans. In *Intelligent Virtual Agents*, pages 1–14. Springer, 2007.

[5] Alex J. Champandard. Applying rapidly-exploring random trees to games. http://aigamedev.com/open/highlights/rapidly-exploring-random-trees/ [online], 2007.

[6] G. Erinc a S. Carpin. A genetic algorithm for nonholonomic motion planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1843–1849. IEEE, 2007.

[7] D.H. Hale, G.M. Youngblood, a P. Dixit. Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.

[8] F.W.P. Heckel, G.M. Youngblood, a D.H. Hale. Influence points for tactical information in navigation meshes. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 79–85. ACM, 2009.

[9] David Šišlák. *Autonomous collision avoidance in air-trafic domain.* PhD thesis, ČVUT, 2010.

[10] F.M. Jönsson. An optimal pathfinder for vehicles in real-world digital terrain maps. http://www.student.nada.kth.se/~f93-maj/pathfinder/contents.html [online], 1997.

[11] S. Koenig a M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2002.

[12] S. Koenig, M. Likhachev, a D. Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[13] S.M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, TR 98-11, Computer Science Dept., Iowa State University, 1998.

[14] S.M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[15] V. Mařík, O. Štěpánková, a J. Lažanský. *Umělá inteligence (1)*. Academia, 1993.

[16] V. Mařík, O. Štěpánková, a J. Lažanský. *Umělá inteligence (3)*. Academia, 2000.

[17] V. Mařík, O. Štěpánková, a J. Lažanský. *Umělá inteligence (5)*. Academia, 2007.

[18] A. Nash, K. Daniel, S. Koenig, a A. Felner. Theta*: Any-Angle Path Planning on Grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 22, pages 1177–1183, 2007.

[19] Alex Nash. Theta*: Any-angle path planning for smoother trajectories in continuous environments. `http://aigamedev.com/open/tutorials/theta-star-any-angle-paths/` [online], 2010.

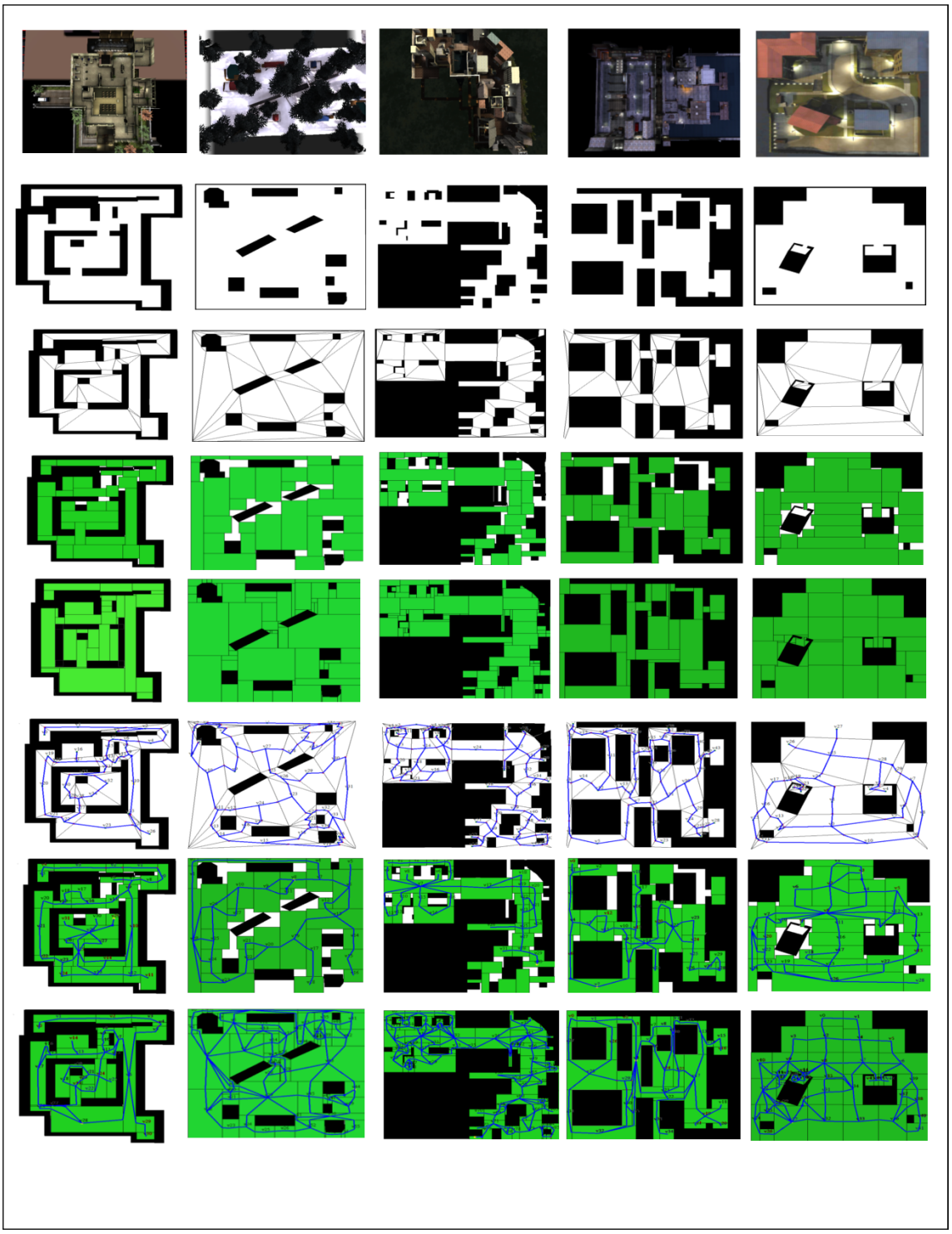
[20] Amit J. Patel. Amit's game programming pages. `http://theory.stanford.edu/~amitp/GameProgramming/` [online], 2011.

[21] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[22] Hani Safaldi. Local path planning using virtual potential field. `http://www.cs.mcgill.ca/~hsafad/robotics/index.html` [online], 2007.

[23] Paul Tozour. Fixing path finding. `http://www.ai-blog.net/archives/000152.html` [online], 2008.

A Diskretizace prostředí pomocí navigačních sítí



Obrázek A.1: V prvním řádku jsou znázorněny obrázky reálného prostředí, druhý řádek znázorňuje dekompozici prostředí pro programové zpracování. Poté následuje popis prostředí pomocí Hertel-Mehlhornova algoritmu, algoritmu SFV a nakonec algoritmu DEACCON. Poslední tři řádky znázorňují vygenerované navigační sítě. Tento obrázek byl převzat z článku [7]

B Obsah CD

Přiložené CD obsahuje zdrojové kódy aplikací použitých pro simulace v poslední kapitole, text této práce v PDF a zdrojové soubory použité pro vytvoření zmíněného PDF systémem LaTeX.

Adresář	Obsah adresáře
`Aplikace`	Zdrojové kódy aplikace
`Latex`	Zdrojové kódy textu práce
`kaslzdenBP.pdf`	Elektronická verze bakalářské práce

Tabulka B.1: Adresářová struktura na přiloženém CD