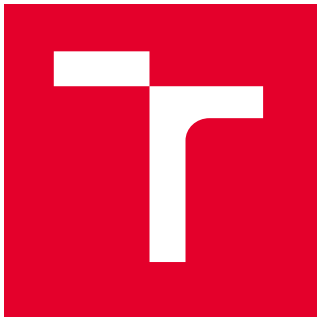


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

HIERARCHICKÉ METODY PLÁNOVÁNÍ CESTY

HIERARCHICAL METHODS OF PATH PLANNING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Veronika Gáčová**

AUTHOR

VEDOUCÍ PRÁCE — **RNDr. Jiří Dvořák, CSc.**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Studentka:	**Bc. Veronika Gáčová**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**RNDr. Jiří Dvořák, CSc.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Hierarchické metody plánování cesty

Stručná charakteristika problematiky úkolu:

Úkolem systému pro plánování cesty je najít cestu z počáteční do cílové pozice tak, aby se agent nedostal do kolize se známými překážkami a aby byla optimalizována nějaká kriteriální funkce. V případě rozsáhlého prostoru je výhodné prostor hierarchizovat do několika úrovní abstrakce, najít cestu v nejvíce abstraktním prostoru a pak ji postupně zpřesňovat v prostorech s nižší úrovní abstrakce. Tento postup se s úspěchem využívá např. v robotice nebo ve videohrách.

Cíle diplomové práce:

1. Analyzovat přístupy k plánování cesty.
2. Implementovat vybrané hierarchické metody plánování cesty.
3. Provést a vyhodnotit ověřovací a srovnávací experimenty.

Seznam doporučené literatury:

ANTIKAINEN, H. Using the hierarchical pathfinding A* algorithm in GIS to find paths through rasters with nonuniform traversal cost. ISPRS International Journal of Geo-Information. vol. 2, 2013, pp. 996-1014.

BOTEA, A. et al. Near optimal hierarchical path-finding (HPA*). Journal of Game Development,vol. 1, 2004, 30 pages.

BRONDANI, J. R. et al. Pathfinding in hierarchical representation of large realistic virtual terrains for simulation systems. Expert Systems with Applications, vol. 138, 2019, 15 pages.

PARK, B. et al. Incremental hierarchical roadmap construction for efficient path planning. ETRI Journal, vol. 40, issue 4, 2018, pp. 458-470.

RAHMANI, V., PELECHANO, N. Improvements to hierarchical pathfinding for navigation meshes. In Proceedings of the Conference on Motion in Games 2017, Barcelona, Spain, 6 pages.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Diplomová práce se zabývá hierarchickými metodami plánování cesty. Teoretická část je věnována reprezentaci vstupního prostředí mapy a popisu základních algoritmů hledání cesty. Dále je zde uvedena charakteristika hierarchického plánování cesty a popis vybraných hierarchických metod. V praktické části byla ve vytvořeném simulačním prostředí provedena implementace vybraných hierarchických metod a jejich následné srovnání a vyhodnocení pomocí experimentů.

ABSTRACT

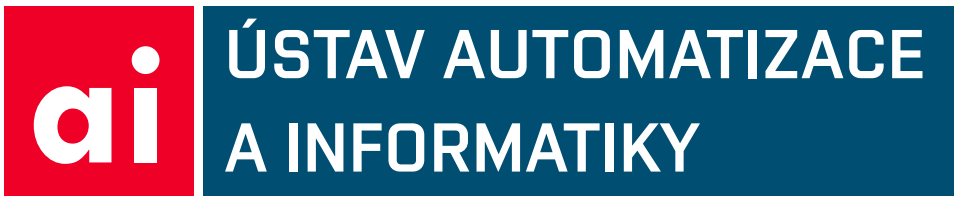
The master thesis deals with hierarchical methods of path planning. The theoretical part is focused on representation of map environment and provides description of path planning algorithms. Further in this thesis description of hierarchical path planning and selected hierarchical methods is provided. In the practical part selected hierarchical methods were implemented. Subsequently in created simulation environment the selected hierarchical methods were compared and evaluated.

KLÍČOVÁ SLOVA

plánování cesty, hierarchické plánování cesty, algoritmus A*, algoritmus HPA*, rozšíření HPA*

KEYWORDS

path planning, hierarchical path planning, A* algorithm, HPA* algorithm, HPA* enhancements



2022

BIBLIOGRAFICKÁ CITACE

GÁČOVÁ, Veronika. *Hierarchické metody plánování cesty*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2022, 74 s. Diplomová práce. Vedoucí práce: RNDr. Jiří Dvořák, CSc.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato diplomová práce je mým původním dílem, vypracovala jsem ji samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 20. 5. 2022 .....

Bc. Veronika Gáčová

PODĚKOVÁNÍ

Na tomto místě chci poděkovat vedoucímu práce RNDr. Jiřímu Dvořákovi, CSc., za odborné vedení, podnětné připomínky a ochotu při psaní mé diplomové práce. Dále bych chtěla poděkovat všem svým blízkým za trpělivost a podporu po celou dobu mého studia.

OBSAH

1 ÚVOD

Plánování cesty je ústředním problémem v řadě aplikací dnešní doby. Své uplatnění nalézá například v GPS navigačních systémech, robotice, logistice nebo počítačových videohrách. Přestože v poslední řadě let došlo ke značnému rozvoji v dané oblasti, v rámci kterého se výrazně zlepšila přesnost a účinnost jednotlivých technik, přitahuje tento problém stále velké množství pozornosti. To souvisí především s rostoucí potřebou prohledávat stále větší a komplexnější prostředí, což je spojeno s rostoucími časovými nároky, spolu se zvyšujícími se požadavky na paměť a výkon výpočetní techniky.

Hierarchické metody plánování cesty představují způsob, jakým lze nároky na vyhledávání minimalizovat. Principem těchto metod je postupná hierarchizace prostředí do několika úrovní abstrakce. Hledání cesty je poté provedeno v prostoru nejvyšší abstraktní úrovně, který umožňuje uvažovat o strategiích plánování cesty z hlediska makro operací. Nalezená abstraktní cesta je poté postupně zpřesňována pro prostory nižších hierarchických úrovní až do prostředí základního rozlišení mapy, kde je nalezena výsledná nízkoúrovňová cesta.

Cílem této práce je analyzovat dosavadní přístupy k plánování cesty, představit hierarchické přístupy spolu s jednotlivými hierarchickými metodami, implementovat vybrané hierarchické metody a na závěr provést vyhodnocovací a srovnávací experimenty.

První kapitola práce se věnuje úvodu do problematiky plánování cesty. Je zde obecně představen problém plánování cesty, následně jsou uvedeny metody zpracování výchozího prostředí mapy, je uveden obecný popis základních vyhledávacích algoritmů a je podrobně popsán *algoritmus A**, který tvoří základ implementovaných metod. V druhé kapitole je popsáno hierarchické plánování cesty spolu s jednotlivými hierarchickými metodami. Následně je uveden podrobný popis implementovaného algoritmu *Hierarchical Pathfinding A** a jeho možných rozšíření *HPA* PS* a *HPA* LEC*. Třetí kapitola se věnuje popisu vytvořeného simulačního prostředí a jednotlivých technik použitých v rámci implementace. Na závěr je uvedena experimentální část práce, ve které byly představeny jednotlivé srovnávací experimenty a provedeno jejich vyhodnocení.

2 PLÁNOVÁNÍ CESTY

Jak bylo zmíněno v úvodu, problém plánování cesty je již několik let stále velkým tématem pro řadu aplikací. A to ať už v rámci průmyslové automatizace, do které spadá oblast robotiky, nebo ve videoherním průmyslu, kdy se stále více investuje do vývoje moderních počítačových her. V současné době je ústředním tématem zpracování stále rozsáhlejších a složitějších prostředí s větším počtem jednotek, které s sebou přináší rostoucí požadavky na paměť a výkon použité techniky a také zvyšující se náklady na samotný čas potřebný pro vyhledávání.

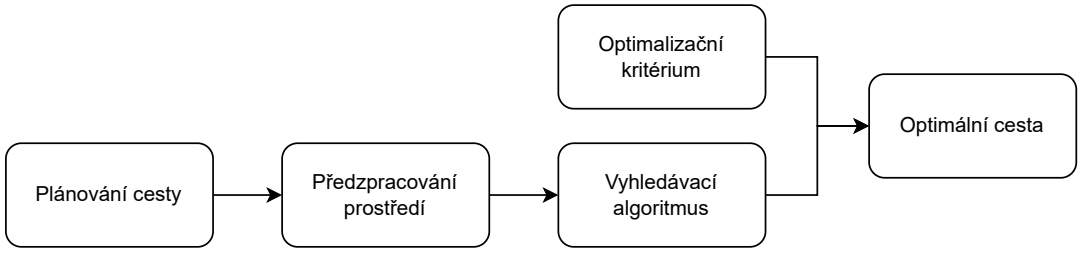
2.1 Plánování cesty obecně

Podstatou plánování cesty agenta je získat posloupnost akcí, které musí agent provést, aby se dostal z počátečního do cílového bodu. Vstupními parametry problému jsou výchozí mapa prostředí, startovací a cílová pozice agenta. Výstupem je postup, kterým se agent dostane z počátečního bodu v mapě do cílového tak, aby nedošlo ke kolizi agenta s neprostupnou oblastí mapy. Neprostupná oblast mapy představuje množinu překážek v mapě. Hledání cesty může mít různá omezení pro různé typy agentů. Například hledáme-li v GPS navigaci pro agenta trasu umožňující jízdu na kole, bude se nalezená trasa pravděpodobně lišit od trasy pro osobní automobil. Každý z agentů zde má jiné vstupní předpoklady a také kritéria, podle kterých je výsledná cesta nalezena, jsou odlišná. Za neprůchozí oblast jsou označována také místa, na kterých se vyskytují ostatní agenti pohybující se po mapě [23, 2].

Problém plánování cesty lze na základě znalostí o vstupní mapě obecně rozdělit do dvou kategorií. První z nich je *globální plánování cesty*, které se označuje jako *deliberativní* a je založeno na kompletní znalosti celého prohledávaného prostředí. Na základě této znalosti je možné vytvořit kompletní globální cestu ještě předtím, než se agent začne pohybovat. Plánování je uskutečněno v offline režimu. Druhým přístupem je *lokální plánování cesty*, které se označuje jako *reaktivní* a jeho podstatou je využití senzorů pro orientaci a získávání informací o prostoru. Agent má v tomto případě neúplnou nebo pouze částečnou znalost o okolním prostředí. Tuto znalost získává online během pohybu, a to za pomocí různých čidel a sensorů, které jsou na něm umístěny. Cílem agenta je následovat cestu k cíli a přitom se vyhnout všem možným překážkám na trase. Lokální vyhledávání má rychlou odezvu ve srovnání s vyhledáváním globálním [9, 41]. Dále se zaměříme především na globální plánování cesty.

Klíčové prvky plánování cesty tvoří předzpracování prostředí, aplikace vyhledávacího algoritmu společně s volbou vhodné heuristické funkce a použití optimalizačního kritéria viz obr. 1. Předzpracování prostředí zahrnuje převod původní

mapové reprezentace prostředí na grafovou strukturu. Problém generování grafu pro danou topologii terénu je považován za základ aplikací robotiky a videoher. Z hlediska teorie grafů jsou primárními problémy nalezení cesty mezi dvěma uzly grafu a získání optimální cesty. Kritéria optimalizace mohou být různá, nejčastěji se setkáme s požadavkem na nalezení nejkratší trasy mezi dvěma body. Jako kritérium však může být uvažována také časová náročnost cesty nebo výsledná cena potřebná k dosažení cíle. Ta může souviset například s množstvím spotřebovaného paliva během cesty, a to v závislosti na typu agenta, rychlosti pohybu nebo vlastnostech terénu [6, 33, 1].



Obr. 1: Proces plánování cesty

Problém nalezení nejkratší cesty je definován jako proces hledání cesty mezi dvěma vrcholy grafu, přičemž je prováděna minimalizace přes součet hodnot hran dané cesty, kde ohodnocení hran odpovídá vzdálenosti mezi jednotlivými uzly. Důležitým faktorem při hledání cesty je nutnost vyhnout se terénním překážkám vyskytujícím se v daném prostředí. K nalezení nejkratší cesty v grafu se využívá algoritmů hledání cesty. Takových algoritmů existuje několik, těmi nejvýznamějšími jsou *A**, *Dijkstrův algoritmus* a *Uniform-Cost Search* (UCS). Mezi nejčastěji používané heuristiky patří *manhattanská*, *euklidovská* a *diagonální* metrika, a dále například metrika *Čebyševova* [23, 39, 33].

2.2 Reprezentace prostředí

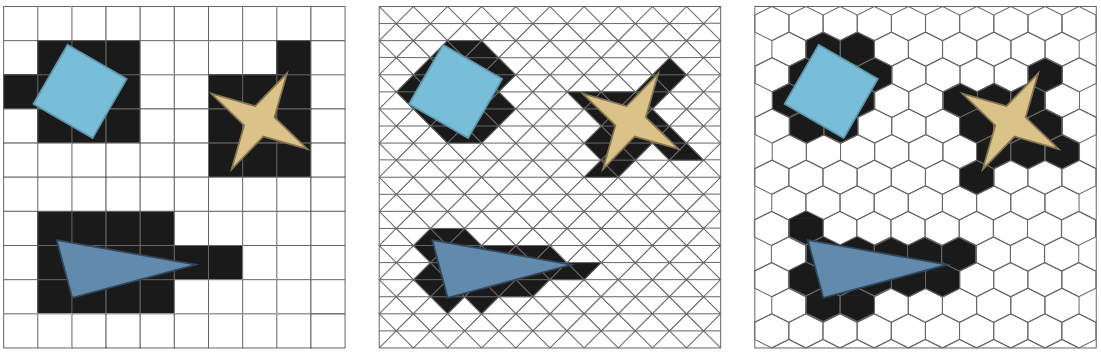
Jedním z klíčových kroků při plánování cesty je proces předzpracování prostředí, tzv. *preprocessing*. To představuje vytvoření odpovídající grafové struktury. Výsledný graf by měl být co nejmenší, aby umožňoval rychlé vyhledávání. Pokud je graf příliš velký, vyhledávání se výrazně zpomalí. Na druhou stranu by diskretizace měla být co nejjemnější, aby oblasti odpovídající uzlům grafu nebyly příliš velké. To by vedlo k aproximační chybě, která končí v suboptimálních cestách [16].

Prostředí je definováno vstupní mapou a lze ho reprezentovat několika způsoby v závislosti na použité metodě. Metody zpracování prostředí můžeme rozdělit do dvou základních kategorií. První z nich jsou metody založené na **rozkladu**

do buněk. Tyto techniky rozkládají původně spojité prostředí na buňky. Každá buňka je obvykle definována konvexním mnohoúhelníkem a představuje oblast prostupného terénu, tedy prostoru bez překážek. Do této kategorie patří rozklad pomocí *pravidelné* a *nepravidelné mřížky* nebo prostřednictvím *navigační sítě*. Výsledkem rozkladu je graf, jehož vrcholy jsou všechny buňky neobsahující překážku a hrany spojují sousedící bunky. Druhou kategorií jsou metody vytvářející cesty mezi body rozmístěnými v modelovaném prostředí, tzv. **metody mapy cest**. Ty, stejně jako u předchozího, převádí původně spojité prostředí na grafovou strukturu reprezentující volný pracovní prostor. Existují tzv. *deterministické* mapy cest, kde body modelového prostředí tvoří vrcholy grafu. Tyto body obvykle korespondují s vrcholy jednotlivých překážek. Hrany grafu představují cesty, po kterých je možné se pohybovat. Těmito metodami rozkladu jsou například *graf viditelnosti* nebo *Voroného diagram* [35, 26, 27]. Další skupinou jsou tzv. *pravděpodobnostní* mapy cest (Probabilistic Roadmaps), kde se náhodně vygenerují body ve volném prostoru a ty se pak spojí hranami neprotínajícími překážky [3]. Výše uvedené metody jsou popsány v následujícím textu.

2.2.1 Pravidelná mřížka

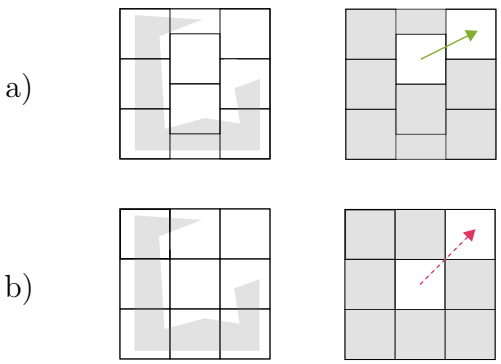
Pravidelná mřížka dělí původní mapu do buněk stejného tvaru a velikosti. Počet buněk a hran nezávisí na počtu překážek v mapě. Každá z buněk je označena jako průchozí nebo blokovaná podle toho, zda se v ní nachází navigační překážka. Grafová reprezentace je tvořena uzly reprezentující volné buňky a hranami, které je propojují. V současnosti jsou používány zejména tři typy buněk, a to čtvercové, trojúhelníkové a šestiúhelníkové viz obr. 2. Pohyb po mřížce je ukutečňován pomocí přesunů z aktuální buňky na některou ze sousedních buněk. V případě čtvercové mřížky je uvažováno 8 možných pohybů (2 horizontální, 2 vertikální, 4 diagonální) [8, 35, 27].



Obr. 2: Pravidelná mřížka (a) čtvercová (b) trojúhelníková (c) šestiúhelníková

Výhodami pravidelných mřížek je jejich snadná implementace, jednoduchost datové struktury a způsob, jakým lze mřížku snadno aktualizovat. Klíčovou nevýhodou je fakt, že mohou vést k podstatně velkým vyhledávacím prostorům, zejména při reprezentaci rozsáhlého virtuálního terénu s vysokým stupněm granularity. To má negativní vliv na výkon a využití paměti systému [8, 35].

Další nevýhodou dané metody je, že nemusí nalézt cestu, přestože taková cesta existuje. To je zapříčiněno aproximací prostředí do buněk stejného tvaru, kdy není uvažován tvar a velikost překážek. Celá buňka tak může být označena za obsazenou, přestože obsahuje překážku, která ji zcela nevyplňuje, nebo do ní jen okrajově zasahuje. Z tohoto důvodu se někdy využívá tzv. posouvání buněk. Na obr. 3 lze vidět, že reprezentace pomocí posunutých buněk je v případě a) schopna nalézt diagonální cestu [27, 26].



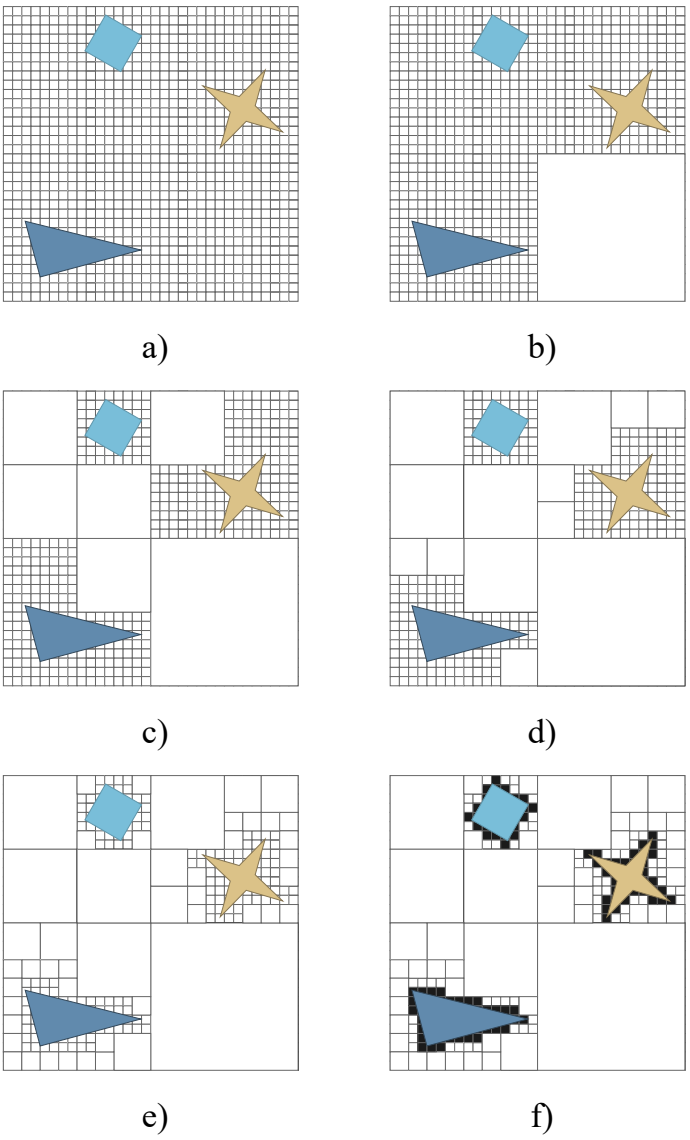
Obr. 3: Odlišná reprezentace prostředí mřížkou s posunutými buňkami (podle [35])

2.2.2 Nepravidelná mřížka

Nepravidelné mřížky se vyznačují buňkami, které mohou mít různé tvary a velikosti. Technikou realizace takových mřížek jsou kvadrantové stromy, neboli *quadtrees*. Klíčovou myšlenkou kvadrantového stromu je rozdělit původní mapu na čtyři podoblasti, jak je znázorněno na obr. 4. Pokud podoblast neobsahuje žádnou překážku, není potřeba ji dále dělit a je brána jako konečná celoprůchozí buňka. Pokud buňka překážku obsahuje, rozdělíme ji stejným způsobem jako původní mapu. Tento postup opakujeme pro každé pole obsahující překážku až do té doby, kdy pracujeme s buňkami základního rozlišení mapy, tj. kdy buňky jsou dále již nedělitelné [8, 35].

Jak lze vidět v případě f), výsledná struktura nepravidelné mřížky obsahuje buňky většího rozlišení v oblasti okolo překážek, naopak v prostředí, kde se žádná překážka nevyskytuje, je reprezentace prostředí hrubší. Tento princip zpracování prostředí přispívá k redukci výsledného počtu uzlů grafové reprezentace, který je potřeba prohledat v rámci algoritmu plánování cesty. Výhodou je tak snížení potřebné paměti pro běh algoritmu ve srovnání s běžnými mřížkami. Nevýhoda dané

metody se projevuje především v mapách s vysokou hustotou překážek, kdy použití kvandrantového stromu ztrácí význam.

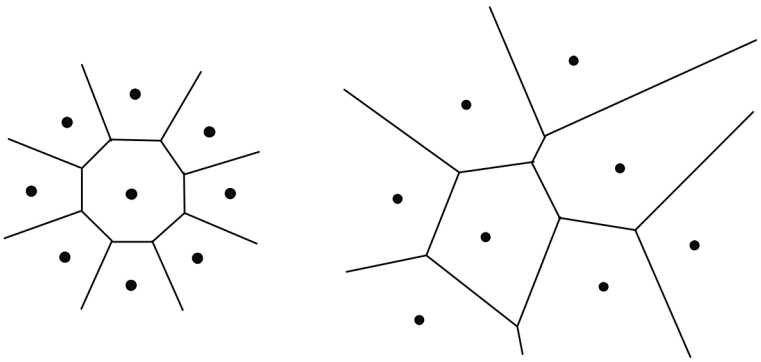


Obr. 4: Konstrukce nepravidelné mřížky - kvadrantový strom

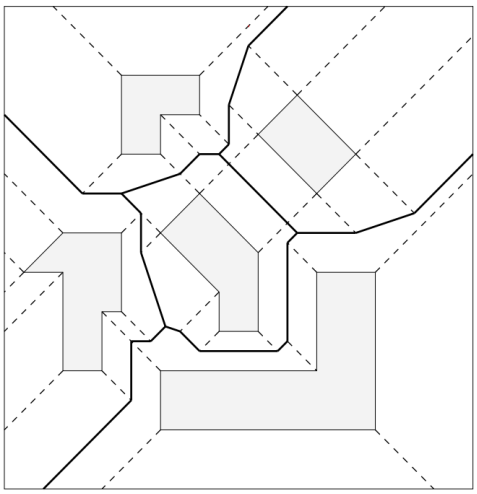
2.2.3 Voroného diagram

Voroného diagram je rovinná geometrická struktura tvořená body, které mají stejnou vzdálenost od dvou nebo více překážek. Kolem každého bodu představujícího překážku je tak vytvořena oblast sdružující body, které jsou k němu blíže než ke všem ostatním. Body stejně vzdálené od dvou různých překážek tvoří hranici mezi dvěma oblastmi. Spojením hraničních bodů vznikají úsečky, které definují hrany diagramu. Po nich je možné se pohybovat po mapě v dostatečné vzdálenosti od překážek a bez rizika kolize s kteroukoli z nich [25, 11, 29].

Grafová struktura je potom tvořena uzly reprezentujícími body se stejnou vzdáleností od tří nebo více překážek a hranami, které odpovídají bodům stejně vzdáleným od překážek dvou [25, 11]. Ukázka Voronoiova diagramu s bodovými překážkami je uvedena na obr. 5. Na obr. 6 lze vidět Voroného diagram pro překážky ve tvaru polygonů.



Obr. 5: Voroného diagram - body [33]

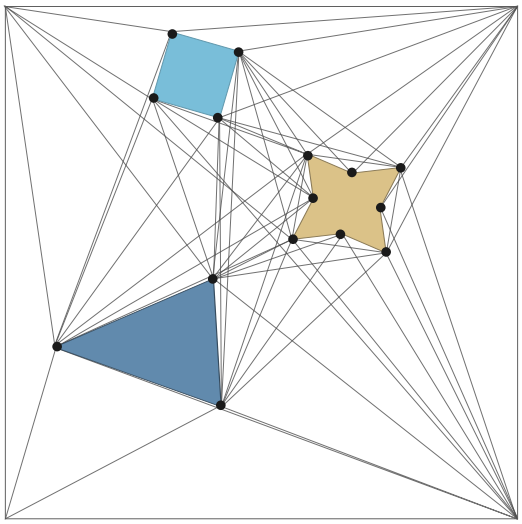


Obr. 6: Voroného diagram - polygony [29]

2.2.4 Graf viditelnosti

Principem grafu viditelnosti je modelovat veškerou topologii prostředí vzájemným propojením vrcholů překážek spolu s připojením počátečního a cílového bodu. Grafová struktura je tvořena uzly reprezentujícími výše zmíněné body a hranami, kterými jsou všechny spojnice uzlů, které neprotínají žádnou z překážek. Propojené uzly jsou tak vzájemně viditelné [8, 35, 27, 26].

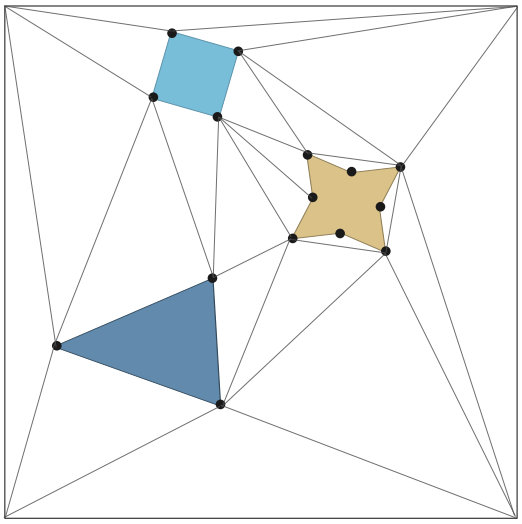
Cesty nalezené v grafu viditelnosti představují skutečné nejkratší cesty. Při prohledávání grafu viditelnosti tak můžeme získat nejkratší možnou trasu z počátečního do cílového bodu. Nejkratší cesta mezi dvěma uzly je buď přímka, nebo tečna k překážkám [8, 35].



Obr. 7: Graf viditelnosti

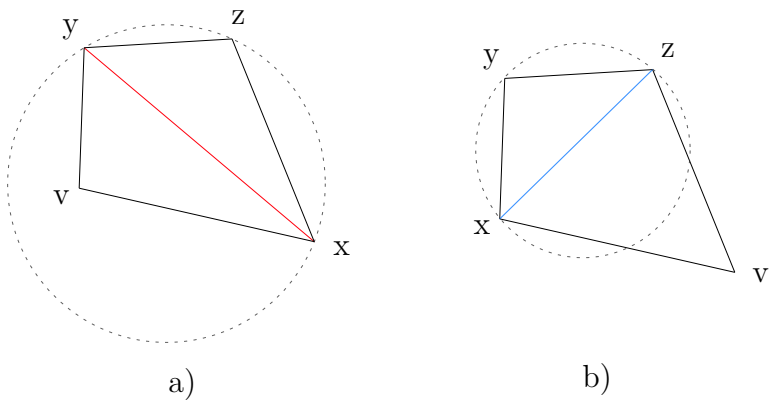
2.2.5 Navigační síť

Navigační síť (ang. *Navigation Mesh*) je obecně složena z konvexních polygonů s libovolným počtem stran, které reprezentují prostupné oblasti mapy, po kterých se agent může pohybovat, tedy prostředí bez překážek. Dále se zaměříme pouze na trojúhelníkové sítě viz obr. 8.

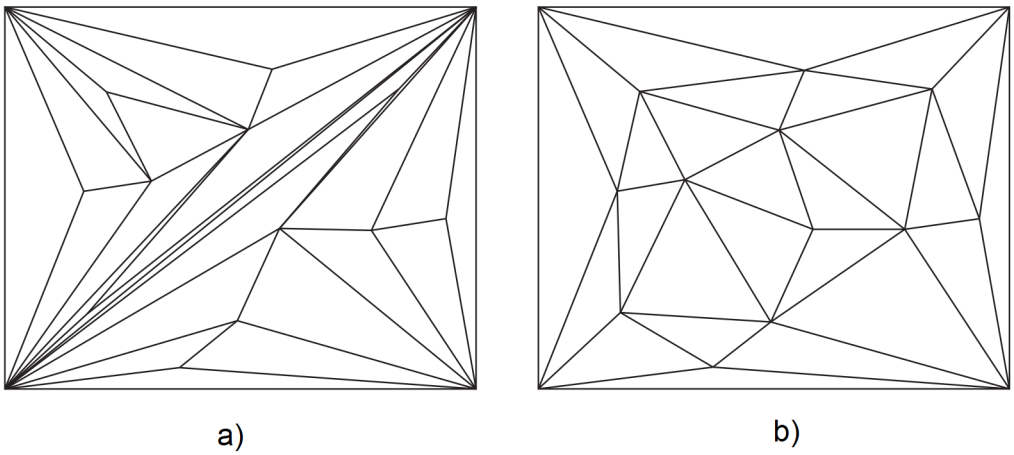


Obr. 8: Navigační síť

Existují různé přístupy, jak provést triangulaci prostředí, jedním z nich je *Delaunayova triangulace*. Jejím cílem je maximalizovat minimální úhel každého trojúhelníku a vytvořit tak síť, kde jsou trojúhelníky co nejrovnoměrnější. Aby byla zajištěna úhlová optimalita, využívá se kružnice opsané trojúhelníku. Pro každý trojúhelník sítě potom platí, že kružnice opsaná danému trojúhelníku je prázdná, tzn. uvnitř kružnice neleží žádný ze zbývajících vrcholů sítě. Pokud opsaná kružnice takový vrchol obsahuje, je provedeno překlopení hrany viz obr. 9. Tím dojde ke změně opsané kružnice, uvnitř které se již nenachází žádné další vrcholy. Daným způsobem lze převést jakoukoli obecnou triangulaci na triangulaci Delaunayovu [8, 35, 26]. Rozdíl v reprezentaci mapy pomocí triangulace před a po překlopení hran je zobrazen na obr. 10.



Obr. 9: Delaunayova triangulace: stav (a) před (b) po překlopení hrany (podle [26])



Obr. 10: (a) Obecná triangulace (b) Delaunayova triangulace [26]

2.3 Základní algoritmy hledání cesty

Druhým krokem v procesu hledání cesty je použití samotného vyhledávacího algoritmu. Jeho úlohou je efektivně nalézt optimální trasu z počátečního do cílového uzlu grafu, který byl vytvořen v předchozím kroku. Uzly grafu představují ve stavovém prostoru množinu všech možných stavů řešeného problému a hrany definují pravidla pro přechod mezi nimi. Pokud je stavový prostor rozsáhlý, je neefektivní prohledávat systematicky všechny jeho stavy (byly by nadbytečně prohledávány i stavy, které nevedou k cíli). Pro efektivní prohledávání stavového prostoru je výhodné využít znalostí o řešeném problému, které mohou prohledávaný stavový značně zredukovat a způsobí, že bude algoritmus postupovat přímočařeji k cíli. Jako znalosti o řešeném problému se často využívá různých heuristik. Z hlediska toho, zda algoritmus při vyhledávání heuristiky využívá či nikoli, lze dělit metody prohledávání grafu na *informované* a *neinformované* [14, 23, 25].

Neinformované metody

Neinformované metody nevyužívají žádné znalosti o stavovém prostoru. Z toho důvodu jsou vhodné pro aplikaci na úlohy s menším počtem možných stavů. Tento způsob se někdy označuje také jako tzv. *slepé prohledávání*. Zástupci těchto metod jsou *prohledávání do šířky* (Breadth-First Search, BFS) a *prohledávání do hloubky* (Depth-First Search, DFS). Dále sem patří například *Bidirectional Search* (BIDI) nebo *Uniform-Cost Search* (UCS). BFS a DFS se liší ve způsobu, jakým expandují jednotlivé uzly grafu. Rozdíl je v pořadí, v jakém jsou uzly vybrány ze seznamu OPEN, který obsahuje dosud neprohledané uzly. U obou algoritmů je v prvním kroku do seznamu OPEN vložen počáteční stav. Ten je následně expandován a umístěn do seznamu expandovaných uzlů CLOSED. Do seznamu OPEN jsou poté přiřazeni jeho následníci (podle zvolené metody buď na začátek, nebo na konec fronty). Daný proces se opakuje až do té doby, dokud jeden z následníků není cílovým stavem, nebo dokud seznam OPEN není prázdný (cesta nebyla nalezena) [14, 23].

Uvažujeme-li hloubku uzlu jako počet hran od počátečního k aktuálně expandovanému uzlu, potom metoda BFS expanduje nejprve uzly s nejmenší hloubkou a metoda DFS naopak ty s hloubkou největší. BFS využívá v rámci seznamu OPEN princip *First-In-First-Out* (FIFO), kdy se nejprve expandují uzly vložené do seznamu jako první. DFS naopak pracuje se seznamem OPEN jako s *Last-In-First-Out* (LIFO) frontou, kde se jako první expandují uzly naposledy vložené. Výhodou BFS je, že nalezne nejkratší cestu, pokud taková existuje. DFS má naopak nižší nároky na paměť, neboť expanduje méně stavů. Nevýhodou DFS je, že nemusí najít optimální cestu a při omezení maximální hloubky nemusí cestu najít vůbec (přestože existuje) [14, 23].

Rozšiřujícími variantami jsou *Depth-Limited search* (DLS) omezující hloubku vyhledávání nebo *Iterative-Deepening search* (IDS), který neustále zvyšuje limit pro hloubku vyhledávání DLS, dokud není nalezeno řešení. Zmíněné obousměrné vyhledávání (BIDI) provádí dvě současná vyhledávání metodou BFS, kdy jedno začíná v počáteční a druhé v cílové pozici. Prohledávání je vedeno směrem ke vzájemnému středu, pokud je zde nalezen společný uzel, existuje cesta mezi kořenovým a cílovým uzlem. Poslední zmíněná metoda UCS je vhodná pro grafy s ohodnocenými hranami. UCS vyhodnocuje graf pomocí prioritní fronty, která je řazena sestupně podle celkové ceny cesty vedoucí ke konkrétnímu uzlu [23].

Informované metody

Informované metody využívají hodnotící funkci pro výběr stavu vhodného k expanzi. Často nepracujeme s exaktními hodnotícími funkcemi, ale s jejich odhady neboli heuristikami. Ty slouží k ohodnocení jednotlivých uzlů, na základě kterého je vybrán uzel k expanzi. Čím kvalitnější hodnotící funkce bude, tím efektivnější bude prohledávání. V takovém případě budou expandovány pouze uzly, které vedou k cíli, a zároveň bude zamezeno prohledávání neperspektivních uzlů. Metody jsou narozdíl od neinformovaných vhodné i pro řešení složitějších úloh ve větším stavovém prostoru [25, 14, 23].

Mezi informované metody patří *gradientní algoritmus* (Hill-Climb search), *algoritmus uspořádaného prohledávání* (Best-First Search) a jeho varianty *hladový algoritmus* (Greedy Best-First Search), *algoritmus paprskového vyhledávání* (Beam Search). Gradientní algoritmus expanduje vždy jako první uzly s nejlepší hodnotou řídící funkce. Algoritmus si nezaznamenává žádné informace o rodičovském uzlu, zpětné prohledávání tak u tohoto algoritmu není nožné. Jeho nevýhodou je možnost uvíznout v lokálním extrému, neboť nejlepší hodnotící funkce uzlu v aktuálním okamžiku nemusí být celkovou nejlepší hodnotou. Algoritmus Best-FS již využívá pamět pro uložení informace o rodiči daného uzlu. Tím je umožněno zpětně prohledat uzly, které sice neměli v minulosti nejlepší hodnotu řídící funkce pro daný okamžik, ale z globálního hlediska mohou nalézt lepší řešení [23, 14].

Mezi v současnosti nejvýznamější metodu této kategorie se řadí *algoritmus A**. Ten je jedním z nejznámějších vyhledávacích algoritmů pro hry a robotiku. Byl prvním algoritmem založeným na Best-First vyhledávání využívajícím heuristickou funkci. Z různých variant algorimu A* vychází řada dalších metod jako například *Iterative Deepening A** (IDA*), *Simplified Memory-Bounded A** (SMA*), *Dynamic A** (D*) a *Anytime Repairing A** (ARA*) [23, 25, 14]. Algoritmus A* bude z důvodu využití v praktické části podrobně popsán v podkapitole 2.4.

2.3.1 Složitost algoritmů

Algoritmy plánování cesty lze porovnávat podle jejich výpočetní složitosti. K definování složitosti algoritmů slouží Big-O notace viz tab. 1. Big-O notace charakterizuje funkce podle rychlosti jejich růstu. Rychlost růstu jednotlivých funkcí je definována pomocí asymptotické horní meze, která vyjadřuje horní hranici odhadu složitosti. Různé funkce se stejnou rychlostí růstu mohou být reprezentovány pomocí stejné Big-O notace. Algoritmy lze hodnotit z hlediska prostorové a časové složitosti. Prostorová složitost zahrnuje míru potřebné paměti během vyhledávání. Časová složitost udává nejhorší možný čas nutný k nalezení řešení. U algoritmů se sleduje také jejich úplnost a optimalilita. Algoritmus je úplný, jestliže pokaždé najde cestu mezi dvěma body, pokud taková cesta existuje. Optimální algoritmus vrací skutečnou nejkratší cestu [23, 33]. Přehled složitosti konkrétních algoritmů je uveden v tab. 2.

Tab. 1: Typické příklady časové složitosti

Big-O Notace	Složitost
$O(1)$	Konstantní
$O(n)$	Lineární
$O(log\ n)$	Logaritmická
$O(n^2)$	Kvadratická
$O(c^n)$	Geometrická
$O(n!)$	Kombinatorická

Tab. 2: Přehled složitosti vybraných algoritmů (podle [23])

Algoritmus	Časová složitost	Prostorová složitost	Optimalita	Úplnost	Vychází z
BFS	$O(b^d)$	$O(b^d)$	Ano	Ano	
BIDI	$O(b^{d/2})$	$O(b^{d/2})$	Ano	Ano	BFS
UCS	$O(b^d)$	$O(b^d)$	Ano	Ano	BFS
DFS	$O(b^m)$	$O(bm)$	Ne	Ne	
DLS	$O(b^l)$	$O(bl)$	Ne	Ne	DFS
IDS	$O(bd)$	$O(bd)$	Ano	Ano	DLS
BestFS	$O(b^m)$	$O(b^m)$	Ne	Ano	BFS/UCS
A*	$O(b^d)$	$O(b^d)$	Ano	Ano	BestFS
IDA*	$O(b^d)$	$O(d)$	Ano	Ano	A*

b - faktor větvení, d - hloubka nalezeného řešení,
m - maximální hloubka grafu, l - limit prohledávané hloubky

2.3.2 Heuristiky

Heuristika představuje orientační pravidlo, které využívá znalosti o prostředí a napomáhá řídit vyhledávání algoritmu směrem k cílové pozici. To má za následek snížení množství prohledaných uzlů a tím i rychlejší nalezení řešení. Heuristika nepředstavuje přesnou vzdálenost z aktuálního bodu do cílového, ale pouze odhad dané vzdálenosti. Je velmi důležité, aby heuristická funkce nenadhodnocovala skutečnou délku nejkratší cesty k cíli. Pokud však heuristická funkce poskytuje odhad, který je menší nebo roven skutečným nákladům, algoritmus vždy najde optimální cestu. Heuristická funkce tradičního A * algoritmu využívá manhattanskou metriku. Následná vylepšení zahrnují metriku euklidovskou, diagonální (octile) a Čebyševovu [39, 21]. Grafická zobrazení těchto funkcí jsou uvedena na obr. 11.

V následujícím textu je pro popis jednotlivých metrik uvažován počáteční bod $S = (x_1,\ y_1)$ a cílový bod $G = (x_2,\ y_2)$ se souřadnicemi $x_1,\ y_1,\ x_2,\ y_2$. Informace v dalším textu jsou čerpány z [25, 39].

Manhattanská metrika vychází z pravoúhlého systému ulic na Manhattanu v New Yorku. Umožňuje pohyb ve 4 směrech (2 horizontální, 2 vertikální). Je definována vztahem:

$$d = \sqrt{|x_1 - x_2| + |y_1 - y_2|} \tag{1}$$

Euklidovská metrika umožňuje pohyb v jakémkoli směru. Využití je možné například v grafu viditelnosti. Vzdálenost je definovaná vztahem:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2}$$

Octile metrika pracuje s 8 možnými směry pohybu (2 horizontální, 2 vertikální, 4 diagonální). Je definována rovnicí:

$$d = max(|x_1 - x_2|, |y_1 - y_2|) + (\sqrt{2} - 1) * min(|x_1 - x_2|, |y_1 - y_2|) \tag{3}$$

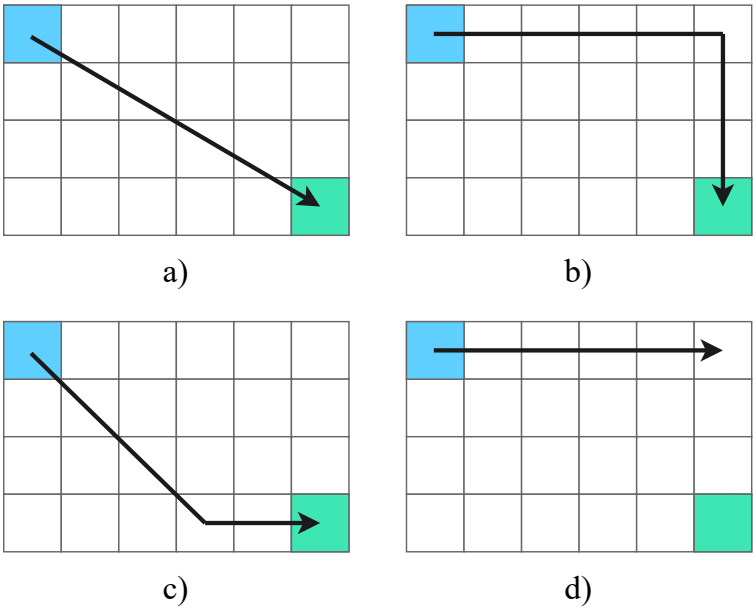
Čebyševova metrika představuje vzdálenost mezi dvěma body jako největší z jejich absolutních hodnot rozdílů podél jakékoliv souřadnicové osy a je vymezená rovnicí:

$$d = max(|x_1 - x_2|, |y_1 - y_2|) \tag{4}$$

V [39] je navržen další typ heuristické funkce, který představuje kombinaci euklidovské a Čebyševovy metriky. Vzdálenost je definována jako:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + max(|x_1 - x_2|, |y_1 - y_2|) \cdot 2 \tag{5}$$

Navržená heuristická funkce spojuje výhody obou metrik. Rozšiřuje schopnost Čebyševovy metriky pro použití v diagonálním směru a zároveň pomáhá kompenzovat nedostatek euklidovské metriky zahrnutím absolutních hodnot. Tento nedostatek spočívá v obtížném rozpoznání vhodného uzlu k expanzi v případě, že je hodnota euklidovské metriky pro několik uzlů velmi podobná [39].



Obr. 11: Metrika (a) euklidovská (b) manhattanská (c) octile (d) Čebyševova

2.4 Algoritmus A*

Algoritmus A* patří mezi informované metody prohledávání, což znamená, že využívá určité heuristiky k nalezení cesty mezi dvěma body. Použití heuristiky vede k prohledání menší části stavového prostoru, a tím i k rychlejšímu nalezení řešení daného problému ve srovnání s metodami neinformovanými. Představen byl v článku *A formal basis for the heuristic determination of minimum cost paths* [18] od trojice autorů P. Hart, N. Nilsson a B. Raphael. Je postavena na principech Dijkstrova algoritmu nejkratší cesty. Narozdíl od něj nehledá nejkratší cestu směrem od počátečního uzlu rovnoměrně ke všem ostatním uzlům v grafu, ale prohledává jenom ty uzly, které mohou vést k cíli. Dosahuje toho právě zavedením heuristické funkce, která v daném kroku pomáhá určit uzel vhodný k expanzi. Nejsou tak zbytečně prohledávány uzly, které nevedou k cíli. Časová složitost algoritmu závisí na způsobu implementace prioritní fronty a také na použité heuristice. V nejhorším případě je počet prozkoumaných uzlů exponenciální vzhledem k celkové délce nalezeného řešení (nejkratší cesta). Časová a prostorová složitost jsou uvedeny v tab. 2. Hodnotící funkce je definována:

$$f(i) = g(i) + h(i), \tag{6}$$

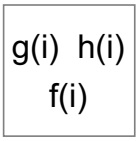
kde $h(i)$ je heuristický odhad vzdálenosti od aktuálního uzlu do cílového a $g(i)$ je dosud zjištěná vzdálenost od počátečního stavu do aktuálního stavu i. Hodnota $g(j)$ se pro každý uzel j spočítá jako

$$g(j) = g(i) + c(i, j), \tag{7}$$

kde $g(i)$ je hodnota této funkce vypočtená pro předchozí (rodičovský) uzel i a $c(i, j)$ představuje vzdálenost přechodu mezi těmito dvěma stavy [27, 14, 23, 21]. K získání heuristického odhadu $h(i)$ používáme metriky popsané v podkapitole 2.3.2.

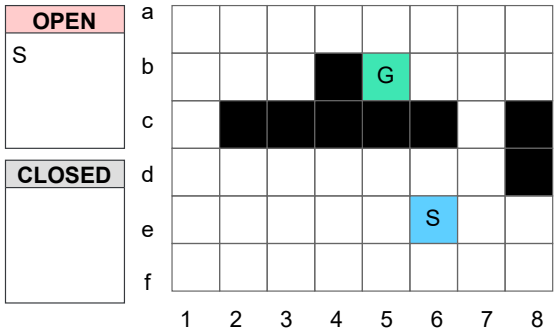
Popis algoritmu

Algoritmus A* pracuje se dvěma seznamy. Prvním z nich je prioritní fronta OPEN, kde se nacházejí uzly čekající na expanzi. Jednotlivé uzly jsou ve frontě uloženy jako čtveřice $\langle stav\ i,\ hodnota\ f(i),\ hodnota\ g(i),\ předchůdce\ stavu\ i \rangle$. Druhým ze seznamů je CLOSED, který obsahuje již expandované uzly. Uzly tohoto seznamu mohou být aktualizovány, najde-li se k nim v průběhu hledání kratší cesta. Algoritmus bude dále demonstrován na mapě reprezentované čtvercovou mřížkou. Na obr. č. 12 je představen systém ohodnocení uzlů, který je dále používán.



Obr. 12: Uzel mřížky - algoritmus A*

Na obr. 13 je zobrazeno prohledávané prostředí v počátečním stavu. Nyní je ve frontě OPEN uložen pouze startovací uzel S s hodnotou funkce $g(i)$ rovnou nule. Ten je následně jako jediný uzel ve frontě vybrán k expanzi. Pokud je startovní uzel zároveň uzlem cílovým, pak algoritmus končí. Pokud tomu tak není, algoritmus pokračuje a daný uzel je expandován.



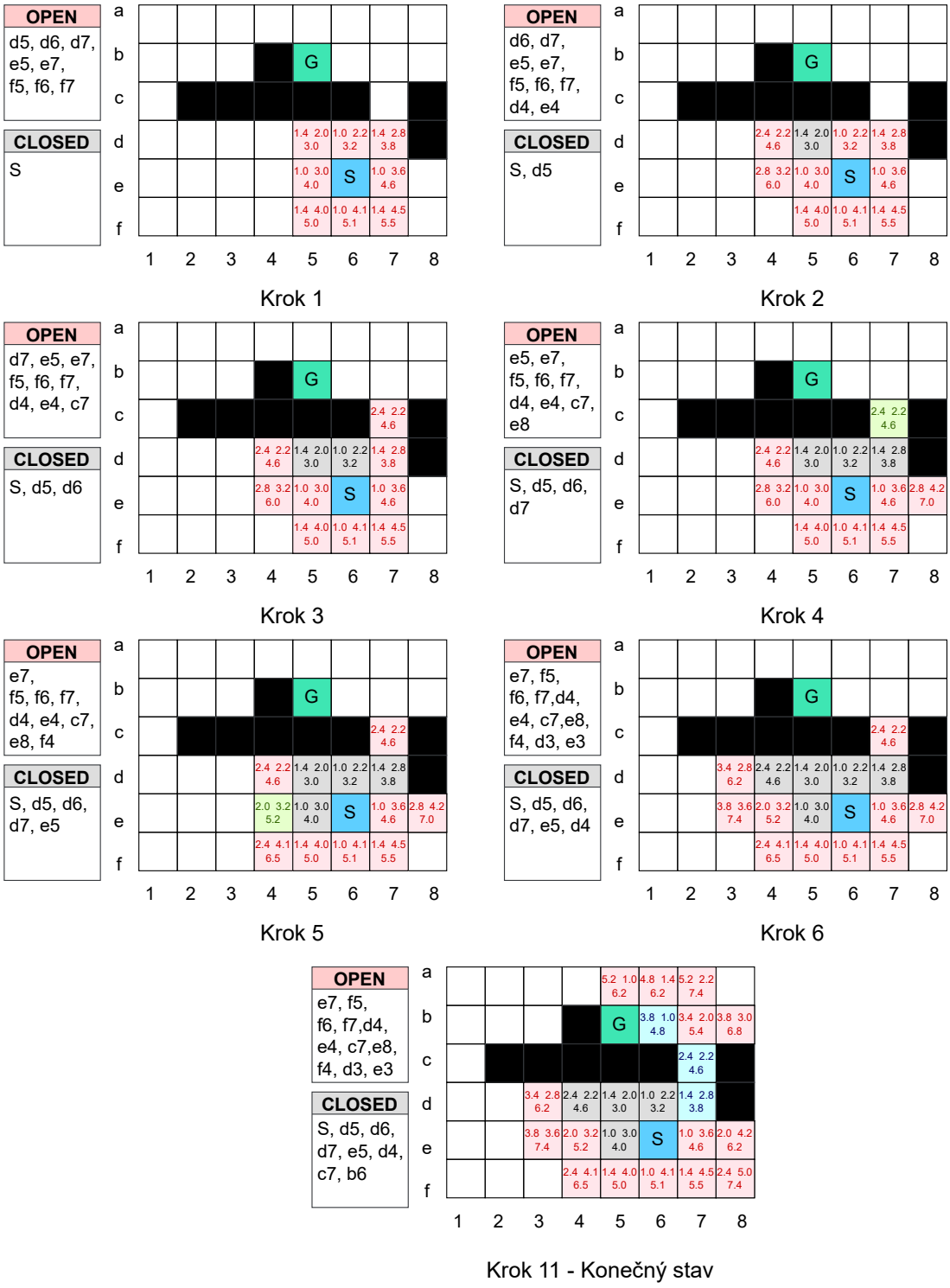
Obr. 13: Testovací prostředí reprezentované čtvercovou mřížkou

Expandovaný uzel S lze vidět v kroku 1 na obr. 14. Do fronty OPEN bylo přidáno jeho 8 sousedících buněk a pro každou z nich byla vypočítána hodnota funkcí $h(i)$, $g(i)$ a následně hodnota funkce $f(i)$. Jelikož byla fronta OPEN prázdná, byly přidány všechny sousední uzly. Pokud by tomu tak nebylo, je nutné nejprve ověřit,

zda se dané uzly ve frontě již nenachází. Pokud ano, provede se srovnání hodnot $f(i)$. Bude-li nově expandovaný uzel mít tuto hodnotu nižší nebo rovnu původnímu, nahradí stávající uzel. Změní se tak hodnoty $g(i)$, $f(i)$ a i příslušný rodičovský uzel (viz zelená pole v krocích 4 a 5). Startovní uzel je následně přesunut do seznamu CLOSED. V kroku 2 je vybrán uzel s nejmenší hodnotou funkce $f(i)$, který není cílovým, a tak je opět expandován. Vypočítají se hodnoty funkcí pro všechny jeho následníky. Ti, kteří se nenacházejí v seznamu OPEN, nebo jsou zde uloženi s horší hodnotou $f(i)$, jsou do seznamu vloženi. Vidíme, že v tomto případě do seznamu nebyly znovu vloženy uzly na pozicích d6 a e5. Prohledaný stav je opět přesunut do seznamu CLOSED a k expanzi je vybrán nový nejslibnější uzel. Tento proces se opakuje až do doby, kdy je z OPEN vybrán stav koncový, nebo do doby, kdy je fronta prázdná. V takovém případě nebyla cesta mezi počátečním a koncovým uzlem nalezena. V našem případě byla výsledná cesta nalezena algoritmem v kroku 11. Konečný stav a výsledná nalezená cesta jsou zde zobrazeny.

Algoritmus 1 *A^* algoritmus*

1: $open := \emptyset$
2: $closed := \emptyset$
3: $g[start] := 0$
4: $h[start] := GetDistance(start, end)$
5: $open \leftarrow (start,\ g[start] + h[start])$
6: **while** $open \neq \emptyset$ **do**
7: $\quad node := open.Best()$
8: $\quad$ **if** $node = goal$ **then**
9: $\quad\quad$ **return** $CreatePath()$
10: $\quad$ **end if**
11: $\quad$ **for each** edge $e = (node, neighbour)$ **do**
12: $\quad\quad gScore := g[node] + e.Weight$
13: $\quad\quad$ **if** $neighbour \in closed$ **then**
14: $\quad\quad\quad$ **continue**
15: $\quad\quad$ **end if**
16: $\quad\quad$ **if** $neighbour \notin open$ **or** $gScore < g[neighbour]$ **then**
17: $\quad\quad\quad g[neighbour] := gScore$
18: $\quad\quad\quad h[neighbour] := GetDistance(neighbour, end)$
19: $\quad\quad\quad open \leftarrow (neighbour,\ g[neighbour] + h[neighbour])$
20: $\quad\quad$ **end if**
21: $\quad$ **end for each**
22: $\quad closed \leftarrow node$
23: **end while**



Obr. 14: Algoritmus A*

3 HIERARCHICKÉ PLÁNOVÁNÍ CESTY

V řadě posledních let výrazně vzrostla potřeba prohledávat stále větší a komplexnější prostředí, a to především v souvislosti s velkým rozvojem v oblasti vývoje navigačních systémů a novodobých videoher. Významným tématem se tak stalo hledání cesty v rozsáhlých vstupních mapách s účelem minimalizovat rozsah prohledávání a také celkovou dobu hledání cesty. Je tedy zapotřebí prohledávanou oblast určitým způsobem zjednodušit a zajistit nalezení požadovaného řešení v co možná nejkratším čase.

3.1 Motivace

S analogií hierarchického hledání cesty se lze běžně setkat v reálném světě. Uvažujeme-li cestování autem na delší vzdálenost, například mezi hlavními městy sousedících států, pak při plánování cesty postupujeme obdobně.

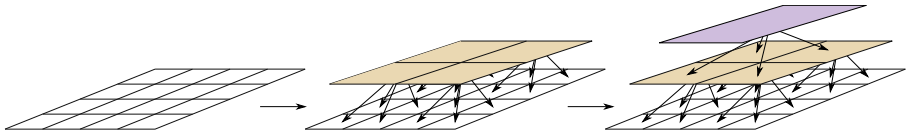
Nejprve hledáme podrobnou cestu z počátečního bodu k okraji města, ve kterém se nacházíme. Typicky se chceme dostat na rychlostní silnici vedoucí do cílového města. Taková nízkoúrovňová cesta bude zahrnovat podrobnou trasu složenou z jednotlivých ulic a silnic v daném městě. Jakmile se dostaneme na jeho hranici, je plánování převedeno na vyšší hierarchickou úroveň. Už nás nezajímají jednotlivé ulice měst, kolem kterých budeme projíždět. Stačí znát pouze seznam měst, případně zemí, přes které hlavní cesta vede. Ulice jsou tedy v rámci hierarchické abstrakce abstrahovány do měst. Různá města jsou pak abstrahována do jednotlivých krajů nebo provincií daného státu. Nejvyšší úroveň abstrakce by potom mohly představovat jednotlivé státy. Dorazíme-li přestřednictvím abstraktní cesty do cílového státu, je opět potřeba přesnější cestovní mapa pro daný stát a následně podrobná mapa pro cílové město. Ve městě opět potřebujeme navigaci na úrovni jednotlivých ulic, abychom se dostali na přesné místo určení [7].

Hierarchie tedy představuje sérii po sobě jdoucích abstrakcí. Z předchozího vyplývá, že pokud projížděná města považujeme za černé skříňky, není zaručeno, že toto hledání najde nejkratší cestu. Trasa přes některá města může být při opuštění dálnice a průjezdu samotným městem kratší. Nalezená hierarchická cesta tedy nemusí být z hlediska kritéria minimální délky optimální [7, 38].

3.2 Hierarchická prostředí

Stěžejním krokem pro aplikaci hierarchických metod je hierarchické zpracování výchozího prostředí. To spočívá ve vytvoření abstraktních levelů, které vzniknou spojením několika uzlů grafu původního prostředí. Abstrakce může být chápána jako

proces snižování rozlišení mapy při zachování informace o vzájemném propojení uzlů původního grafu. Jednoduchý příklad abstrakce je uveden na obr. 15. Hierarchické zpracování se provádí především na mřížkových mapách. Jiné typy map, jako například navigační sítě, se pro vytvoření hierarchie používají také, avšak jejich zpracování je značně komplikovanější [36, 38].

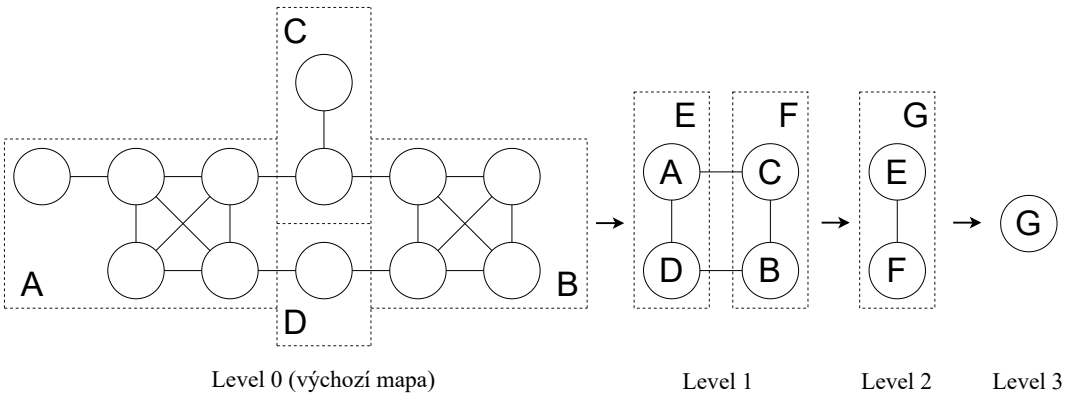


a) b) c)

Obr. 15: (a) Základní rozlišení mapy (b) Level abstrakce 1
(c) Level abstrakce 2 (podle [36])

Abstrakci na mřížkových mapách lze provést primárně dvěma způsoby. Jedním z nich je abstrakce na základě prostorového (sektorového) uspořádání, kterou využívá například algoritmus *Hierarchical Pathfinding A** (HPA*). Sektorové uspořádání rozděluje původní mapu na pravidelné úseky o stejné velikosti. Všechny uzly grafu, které spadají do jednoho z vyčleněných úseků, následně tvoří jeden nový uzel ve vyšší abstrakci. Definované oblasti mají mezi sebou pevně určené body přechodu. Tento způsob je vhodný pro předvýpočet velkého množství menších řešení, která následně urychlují výsledný výpočet prováděný na nejvyšším levelu abstrakce. Abstrakce prostředí na základě sektorového uspořádání bude z důvodu její implementace v praktické části detailně popsána v podkapitole 3.4 [36, 38].

Druhým způsobem je abstrahování prostředí na základě lokálních vlastností uzlů grafu. Důležitý je vzájemný vztah mezi jednotlivými uzly. Tato technika byla poprvé uvedena v článku *Hierarchical A*: Searching Abstraction Hierarchies Efficiently* [20], který diskutoval o seskupování uzlů se svými sousedy. Daný způsob pracuje s pojmy *klika* a *sirotek*. Na základě nich definuje uzly, které budou v rámci nově vznikajícího abstraktního levelu tvořit jeden celek, tedy jeden společný vrchol ve vyšší abstrakci. V teorii grafů se klikou rozumí podmnožina neorientovaného grafu, kde je každý pár uzlů v podgrafu spojen hranou. Sirotek je uzel, který je se zbytkem grafu spojený pouze jednou hranou. V grafové teorii jej nazýváme listem grafu. Tvorba jednotlivých levelů abstrakce pomocí dané techniky je zobrazena na obr. 16. Vidíme, že sirotek je vždy zahrnut do kliky, se kterou sdílí své jediné propojení [36, 38].



Obr. 16: Abstrakce prostřednictvím metody založené na klikách (podle [36, 38])

3.3 Hierarchické metody plánování cesty

Pro plánování cesty v hierarchickém prostředí se využívá zejména algoritmus A*. Na něm jsou založeny již zmíněné hierarchické metody Hierarchical Pathfinding A* (HPA*) a Hierarchical A* (HA*). Dálšími metodami jsou například *Hierarchical Annotated A** (HAA*), *Partial Refinement A** (PRA*), *Triangulation Reduction A** (TRA*) a *Hierarchical NavMesh Path-finding algorithm* (HNA*) [38].

HPA*

Metoda HPA* využívá sektorovou abstrakci prostředí. Základem této techniky je tzv. *online vyhledávání*, které probíhá ve třech krocích. Prvním krokem je provést vyhledávání od počáteční polohy k hranici oblasti, ve které se nachází. Následně je prohledán vygenerovaný abstraktní graf na nejvyšší úrovni abstrakce. V grafu je pomocí algoritmu A* nalezena cesta až k hranici oblasti, ve které se nachází cílový bod. Nakonec se uvnitř cílové oblasti opět provede lokální vyhledávání na nejnižší úrovni směrem k cílové pozici. Možná vylepšení algoritmu HPA* zahrnují vyhlazení suboptimální cesty, použití Dijkstrova algoritmu nebo návrh líného výpočetního schématu (varianta *lazy*) k využití v dynamických prostředích. Algoritmus HPA* je dále detailně popsán v podkapitole 3.4 [7, 38].

HAA*

Technika HAA* je rozšířením HPA* pro agenty různých velikostí a pohybových schopností. Pohybové schopnosti jsou zkoumány především v souvislosti s různým typem terénu v mapě. Tato metoda byla vytvořena především pro využití v moderních počítačových videohrách, kde se často vyskytuje velké množství heterogenních agentů. Je zde zaveden pojem *clearance*, který definuje horní hranici přípustné velikosti agenta a jeho minimální schopnosti pro každé pole mřížky. Pokud dané

parametry nesplňuje, pole se označí jako blokované a agent na něj nesmí vstoupit. Tyto anotace jsou začleněny do A* vyhledávání jako další parametry [38, 17].

HA*

Metoda HA* využívá k vytvoření abstrakce techniku STAR. Ta pracuje na podobném principu jako abstrahování pomocí klik. V technice STAR je uzel s nejvyšším stupněm ve výchozím grafu abstrahován se svými sousedy do jediného abstraktního uzlu vyšší úrovně. Tento postup se opakuje dokud nejsou do abstraktního grafu namapovány všechny uzly původního grafu. Hierarchie se vytváří opakováním procesu na vytvořeném abstraktním grafu, čímž vzniká abstrakce vyšší úrovně. Abstrakce je prováděna až do doby, kdy je původní graf sbalen do jediného abstraktního uzlu. Běh algoritmu je podobný A*. Když je stav expandován, heuristické náklady následovníků se vypočítají jako náklady od tohoto stavu ke stavu cílovému. Výpočet probíhá v levelu abstrakce, ve kterém se expandovaný stav nachází. Tato hodnota, pokud je definována, je následně předána na nižší úroveň abstrakce jako heuristický odhad [38, 20].

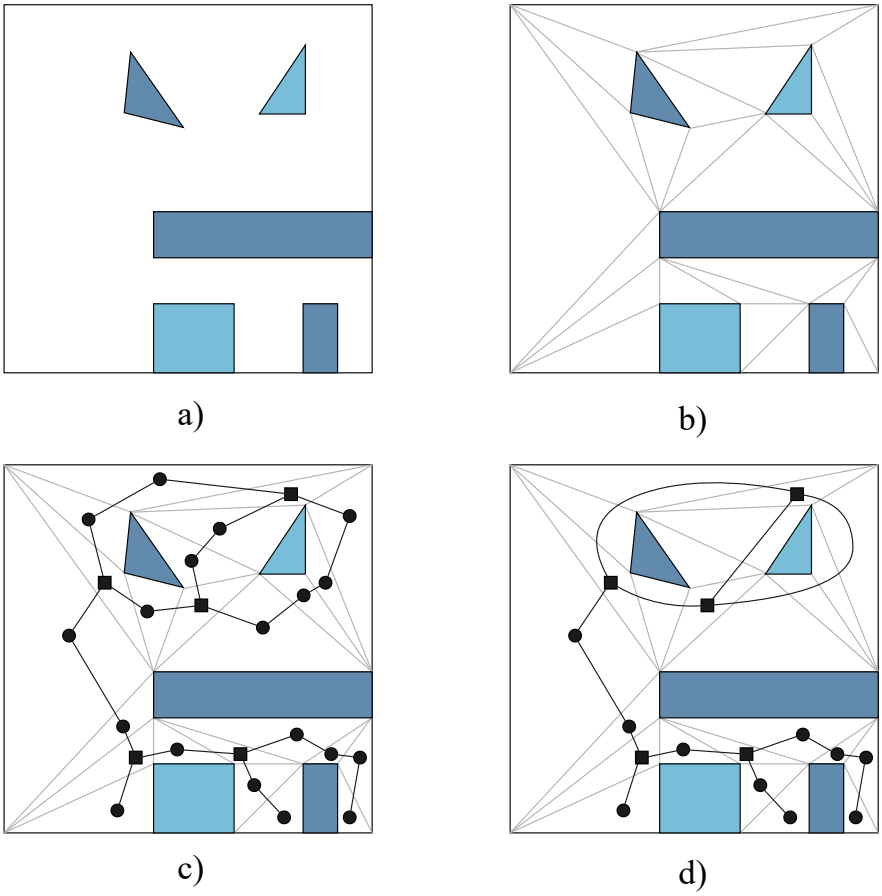
PRA*

Algoritmus PRA* prokládá plánování cesty a samotné provádění (pohyb po cestě). To je umožněno na základě průběžného výpočtu dílčích řešení. Je tak vhodný pro hledání cesty v reálném čase. Tvorba hierarchie je založená klikové abstrakci. Samotný algoritmus je založen na přístupu *QuickPath*. QuickPath funguje tak, že prochází hierarchickým prostředím směrem od nejmenší úrovně abstrakce k vyšším a přitom hledá nejnižší abstraktní uzel, který obsahuje jak počáteční, tak i cílový bod. Cesta mezi těmito dvěma body na úrovni abstrakce je vždy tvořena hranami nižšího levelu mezi danými body. Bylo navrženo několik vylepšení tohoto algoritmu, kterými se samotný PRA* odlišuje od QuickPath. Většinou se jedná o vylepšení toho, jak přesnou vrací cestu pro nižší úroveň abstrakce [38, 36].

TRA*

V rámci metody TRA* je výchozí prostředí zpracováno prostřednictvím Delauneyho triangulace na prostředí složené ze sady trojúhelníků. Tyto trojúhelníky představují prostupnou oblast mapy. V TRA* je každý trojúhelník namapován na uzel v abstraktním grafu. Hrany mezi uzly jsou poté vytvořeny všude tam, kde mezi sebou příslušné trojúhelníky nižší úrovně sdílejí společnou hranu. Jednotlivé uzly jsou rozděleny na stupně v rozmezí 0 až 3 v závislosti na počtu sousedních grafových struktur. Na základě této kategorizace je následně provedena abstrakce výchozího trojúhelníkového prostředí. Příklad takové abstrakce je uveden na obr. 17 [38, 13].

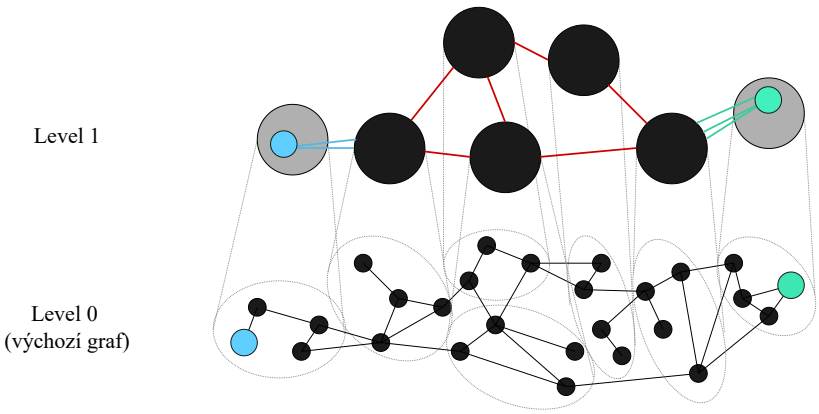
Algoritmus TRA* následně najde cesty ve vytvořené triangulační abstrakci. Na závěr je aplikován přidružený algoritmus TA*. Ten je určen k hledání cesty přes řadu spojených trojúhelníků a vytvoří již konečnou nízkoúrovňovou cestu [38, 13].



Obr. 17: (a) Výchozí prostředí (b) Triangulace prostředí
(c) Triangulační graf (d) Abstrakce triangulačního grafu (podle [13])

HNA*

Vstupní prostředí je v rámci této metody reprezentováno navigační sítí, která je složená z konvexních polygonů. Tato metoda zpracování prostředí se velmi často využívá například v počítačových videohrách. Jednotlivé polygony tvoří uzly základního grafu, který považován za nejnižší úroveň hierarchie (level 0). Tvorba hierarchického prostředí, stejně jako u HPA*, spočívá v seskupování buněk pro vytvoření vyšší abstraktní úrovně. Využívá k tomu víceúrovňový rozdělovací algoritmus *k-way* (MLkP) [24] s informací o dílčích cestách. Algoritmus MLkP začíná fází tzv. *zhrubnutí*, která spočívá v postupné redukci vstupního grafu. Každý graf vyšší úrovně je vytvořen z předchozího spojením co největšího množství sousedních uzlů. Následně je provedena fáze *zjemnění* nalezeného grafu nejvyšší úrovně. Algoritmus pro hierarchické hledání cesty je koncepčně podobný HPA*, ale byl přizpůsoben prostředí polygonálních navigačních sítí. Hledání cesty lze pomocí HNA* provést v na libovolné hierarchické úrovni. Abstrakce grafu reprezentujícího polygonálního prostředí je uvedena na obr. 18. Vyhledávací algoritmus HNA* je spuštěn na úrovni levelu 1 [31, 32].



Obr. 18: Graf prostředí polygonální navigační sítě a jeho abstrakce (podle [31])

3.4 HPA*

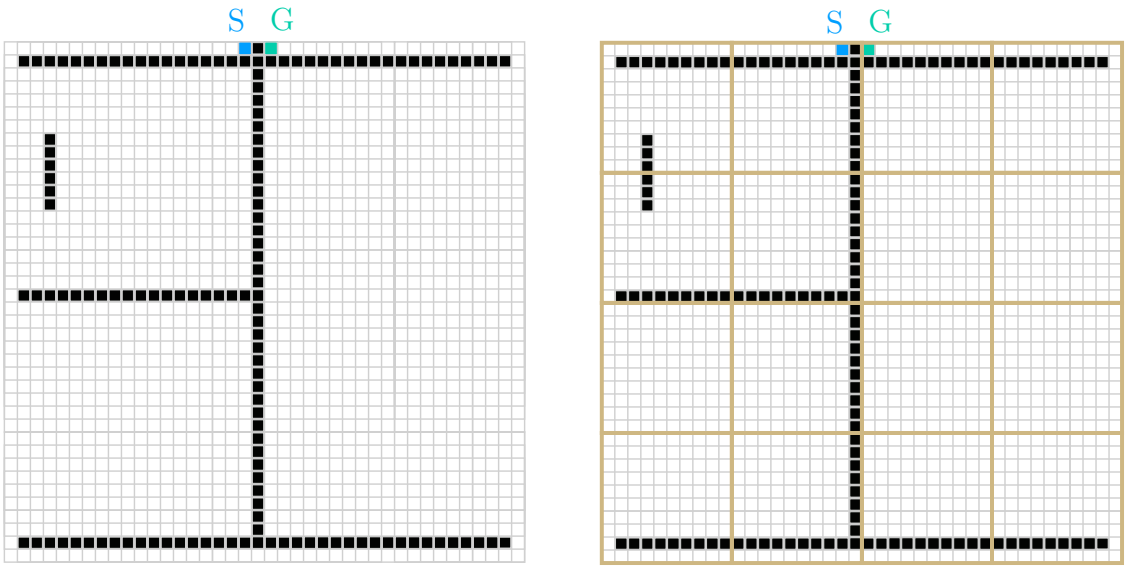
HPA* neboli Hierachical Path-Finding A* je hierarchická metoda plánování cesty založená na hierarchické abstrakci komplexního prostředí mapy, ve kterém je následně hledána suboptimální cesta. Tuto metodu zavedli Adi Botea, Martin Müller a Jonathan Schaeffer ve své práci *Near optimal hierarchical path-finding (HPA*)* [7]. Využívá se ke snížení složitosti hledání cest v rozsáhlých mapách, které mají strukturu mřížky.

V počáteční fázi je původní mapa abstrahována do tzv. *cluster* (shluků) o předem definované velikosti. Shluky jsou oblasti čtvercového nebo obdélníkového tvaru, které v závislosti na jejich velikosti pokrývají určité území původní mapy. Jednotlivé shluky na sebe navazují a výsledná hierarchická struktura pokrývá celou plochu mapy [7].

Hledání hierarchické cesty probíhá ve třech krocích. Nejprve je určena cesta z počátečního bodu k hranici oblasti, ve které se nachází. Poté je nalezena cesta od hranice počáteční oblasti k hranici cílové oblasti. To se provádí na abstraktní úrovni, kde je vyhledávání jednodušší a rychlejší. Vyhledávání probíhá přes relativně velkou oblast, aniž by bylo nutné zabývat se jejími detaily. Nakonec je definována cesta od hranice cílové oblasti do koncového bodu [7].

3.4.1 Tvorba hierarchického prostředí

HPA* využívá sektorovou abstrakci prostředí popsanou v kap. 3.2. Na obr. 19 je zobrazena základní mapa o velikosti 40 x 40, která je následně rozdělena do 16 shluků velikosti 10 x 10. Nově vzniklé shluky jsou v pravé části obrázku vyznačeny hnědými liniemi a označujeme je jako shluky levelu 1. Startovní bod je zde znázorněn



a) b)

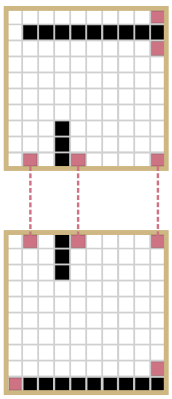
Obr. 19: (a) Základní mapa (b) Abstrakce mapy na úrovni levelu 1 (podle [7])

modrým políčkem, cílový bod je zobrazen zeleně. Černá pole značí překážky, tedy neprůchozí body mapy.

Pro každou linii, která tvoří hranici mezi dvěma shluky identifikujeme (možná prázdnou) množinu vstupů, které je propojují. Každý jeden vstup je tvořen vždy nejdelším možným úsekem bez překážek podél společné hranice obou sousedních shluků. Vstup e je sada polí, u níž platí:

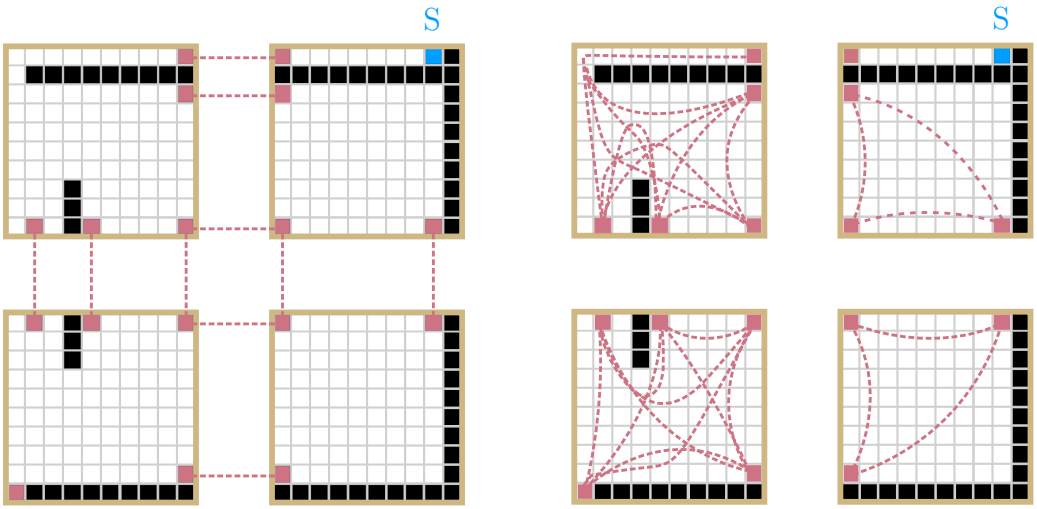
1. Podmínka omezení hranice: Vstup je definován vždy v rámci jednoho shluku, a to podél společné hranice dvou sousedních shluků, přičemž tuto hranici nesmí překročit.
2. Podmínka symetrie: Dvě sousední sady polí tvořící vstup jsou navzájem symetrické.
3. Podmínka volných polí: Vstup nesmí obsahovat pole s překážkou.
4. Podmínka maximalizace: Dokud jsou předchozí podmínky splněny, je vstup postupně rozšiřován.

Na obr. 20 jsou detailně zobrazeny dva sousední shluky z levé horní části ukázkové mapy a způsob, jakým jsou podle [7] identifikovány jejich vstupy. Shluky jsou spojeny celkem dvěma vstupy o šířce 3 a 6 polí v daném pořadí. V závislosti na jejich šířce definujeme pro každý z nich jeden nebo dva body přechodu. Pokud je šířka vstupu menší než předdefinovaná konstanta, v [7] je tato kontansta stanovena na hodnotu 6, pak je definován pouze jeden přechod uprostřed takového vstupu. V opačném případě definujeme dva přechody, každý na opačném konci vstupu. Místa přechodu jsou na obrázku zobrazena jako pole červené barvy spojené přerušovanou čarou s příslušnými poli sousedního shluku.



Obr. 20: Inter-hrany

Jelikož se jedná o hrany spojující dva sousední shluky, mluvíme o tzv. *inter-hranách*. Pro pohyb uvnitř shluku je nutné definovat také jeho vnitřní hrany. Pro každý pár vstupů uvnitř jednoho shluku definujeme hranu, která tyto dva vstupy propojuje. Takto vzniklé spoje nazveme *intra-hranami*. Na obr. 21 můžeme vidět všechny zmíněné hrany vytvořené pro levou horní čtvrtinu původní mapy.



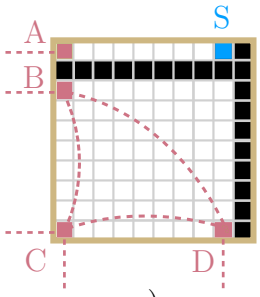
Obr. 21: (a) Inter-hrany (b) Intra-hrany

Ocenění hran je určeno následujícím způsobem:

1. Všechny inter-hrany mají hodnotu přechodu rovnu 1.
2. Délka intra-hrany se počítá hledáním optimální cesty uvnitř shluku. Přímý přechod mezi poli má hodnotu 1, diagonální má hodnotu 1,42.

Kompletní sada hran pro samostatný shluk je znázorněna na obr. 22 a). Doplňující tabulka b) uvádí hodnoty vzdáleností pro každé dva body přechodu uvnitř shluku. Pokud cesta mezi dvěma body neexistuje, tzn. v rámci daného shluku neexistuje

linie, která by tyto dva body spojovala, pak je hodnota jejich vzdálenosti označena jako nekonečno.



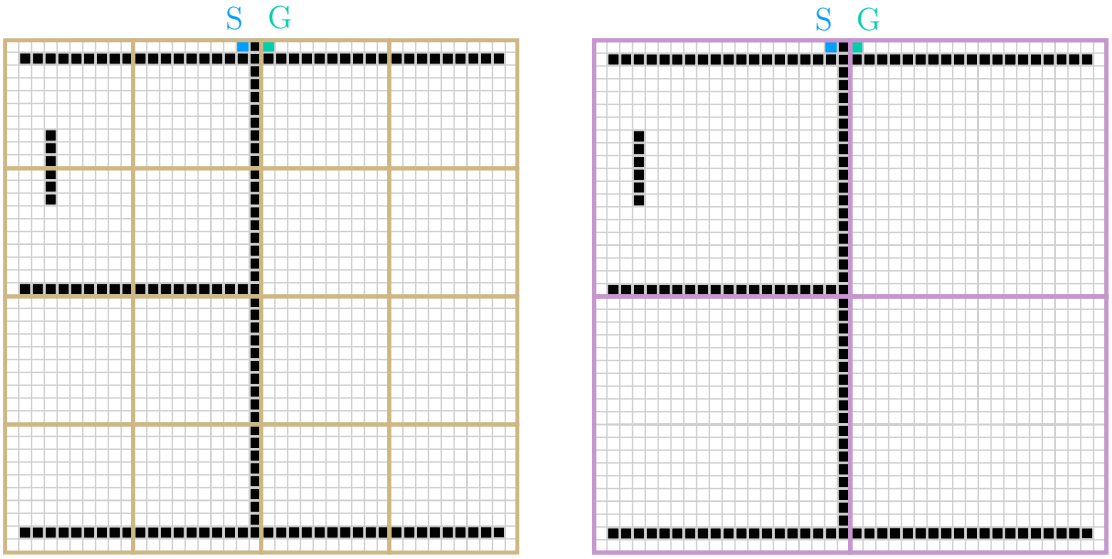
a)

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

b)

Obr. 22: (a) Shluk úrovně 1 se všemi hranami
(b) Ohodnocení vnitřních cest (podle [7])

Vytvoření abstrakce mapy na úrovni levelu 2 je zobrazeno na obr. 23. Každý shluk levelu 2 obsahuje 4 shluky předchozí abstrakce. Jeho strany jsou dvakrát větší, tedy 20 x 20 polí původní mapy, a počet shluků pro tento level se zredukoval na 4. Pro daný případ je tato úroveň konečná, neboť další abstrakcí bychom získali již pouze jeden jediný shluk[1].



a) b)

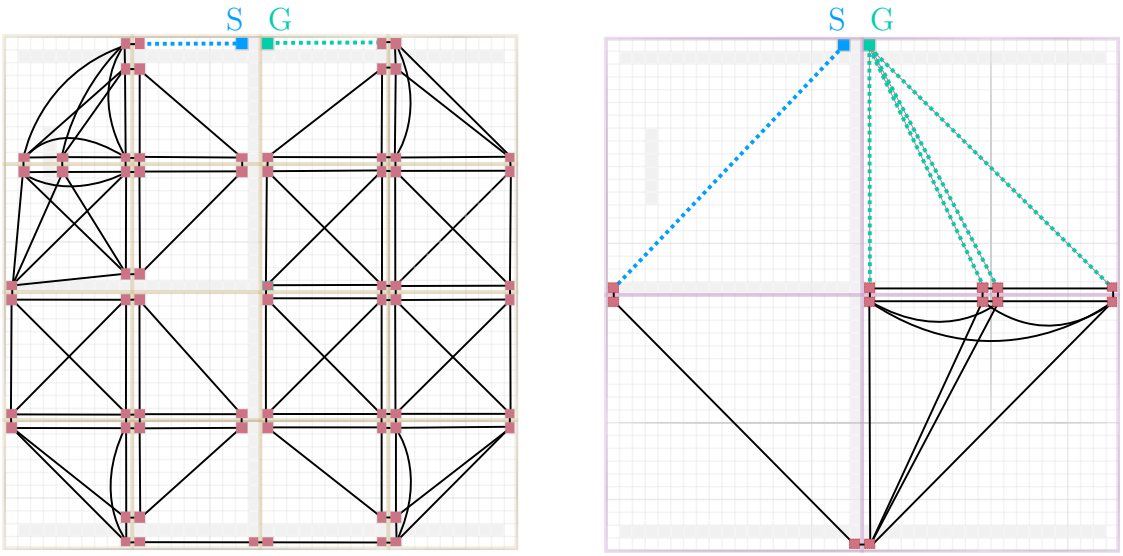
Obr. 23: Abstrakce mapy na úrovni (a) levelu 1 (b) levelu 2

Výslednou grafovou reprezentaci lze vidět na obr. 24. Abstraktní graf je tvořen strukturou vzájemně propojených inter a intra-hran vytvořených dle předchozího. K této struktuře byly následně připojeny počáteční a cílový uzel. Princip jejich vložení je popsán v následující podkapitole. Pro level 1 má graf celkem 68 uzlů,

[1] v případě jednoho shluku neexistují žádná místa přechodu, nelze zde vytvořit cestu

včetně S a G, které se mohou při každém vyhledávání měnit. Na této úrovni abstrakce existuje 16 shluků se 43 vzájemnými propojeními (inter-hrany) a 88 vnitřními spojeními (intra-hrany). Pro level 2 je počet uzlů ještě o poznání nižší, a to 14 uzlů (včetně S a G), dále 4 shluky s celkem 6 inter a 15 intra-hranami. Pro srovnání, nízkoúrovňový (neabstrahovaný) graf obsahuje 1 463 uzlů a 2 714 hran.

Po sestavení abstraktního grafu a nalezení vnitřních cest uvnitř každého shluku, je mapa připravena k použití v rámci hierarchického vyhledávání.



Obr. 24: Grafová reprezentace (a) levelu 1 (b) levelu 2 (podle [7])

3.4.2 Hledání abstraktní cesty

Aby bylo možné hledat cestu v abstraktním grafu, musí být S a G součástí grafu. Uzel S je vložen do abstraktního grafu pro všechny dostupné úrovně (levely) grafu. Vložení představuje nalezení nízkoúrovňové cesty od daného uzlu S ke všem ostatním uzlům daného shluku a následný výběr optimální cesty (zde nejkratší). Mezi počátečním uzlem a vybraným hraničním uzlem je vytvořena hrana s váhou rovnou délce cesty (viz modrá přerušovaná linie na obr. 24). Tato hrana je přidána do abstraktního grafu spolu s uzlem S.

První fáze hledání abstraktní cesty tedy zahrnuje spojení počáteční pozice S s hranicí shluku, ve kterém se nachází. Obdobně je do jednotlivých grafových reprezentací vložen a připojen také koncový uzel G. V dalším kroku je provedeno hierarchické hledání cesty prostřednictvím algoritmu A*. Výsledkem je kompletní abstraktní cesta od počátečního do koncového uzlu.

Tato cesta se skládá ze 3 částí:

1. Reálná cesta (sekvence polí v základním nízkoúrovňovém grafu) od uzlu S k hranici shluku, ve kterém se nachází.

2. Abstraktní cesta na nejvyšší úrovni hierarchie (v daném případě se jedná o level 2) mezi hranicí shluku obsahujícího S k hranici shluku obsahujícího G.
3. Reálná cesta od hranice shluku obsahujícího G k danému koncovému uzlu G.

Po nalezení hierarchické cesty je možné danou cestu zpřesnit a vyhladit. Tyto dodatečné úpravy nazýváme *path-refinement* a *path-smoothing*. Path-refinement, neboli zjemnění nalezené cesty, se využívá k převodu abstraktní cesty na konkrétní cestu v původní mapě. Výsledná cesta je kompletně nízkoúrovňová a je dána přesnou sekvencí původních polí vstupní mapy. Path-smoothing, neboli vyhlazení cesty, je možné použít ke zlepšení kvality dané nízkoúrovňové cesty. Pomáhá optimalizovat nalezené suboptimální řešení, které vzniklo v důsledku definování pouze jednoho přechodového bodu v rámci jinak souvislé linie polí tvořící vstup na hranici každého shluku. Řešení je tak optimální v abstraktním grafu, ale ne nutně v grafu počátečního problému.

V rámci path-refinement nemusí proběhnout hledání optimálních cest v jednotlivých shlucích znovu. Pokud byly při vytváření hierarchické reprezentace jednotlivé optimální cesty nižších úrovní průběžně ukládány, představuje path-refinement pouze tabulkové prohledávání již vypočtených vzdáleností. Pokud k uložení nedošlo, hledají se v jednotlivých shlucích optimální cesty po vzoru nalezené hierarchické cesty vyšší úrovně.

3.5 Rozšiřující metody pro HPA*

V této podkapitole jsou navrženy dva algoritmy hierarchického hledání cesty. První algoritmus *HPA* Post Smoothing* (HPA* PS) využívá metodu HPA* ke zpracování prostředí a nalezení výsledné cesty. Od klasického algoritmu HPA* se liší tím, že upravuje nalezenou cestu tak, aby byla co možná nejkratší. To je uskutečňováno postupným vyhlazováním dané cesty všude tam, kde je to možné. Druhý algoritmus *HPA* Lazy Edge Computing* (HPA* LEC) vychází též z HPA*, avšak na rozdíl od něj nevytváří celé hierarchické prostředí na počátku vyhledávání. Toto prostředí je postupně sestavováno až za běhu algoritmu v závislosti na tom, kudy je směřováno samotné vyhledávání. Algoritmus tak zbytečně nevytváří hierarchii v oblastech, které nevedou k nalezení výsledné cesty.

3.5.1 HPA* PS

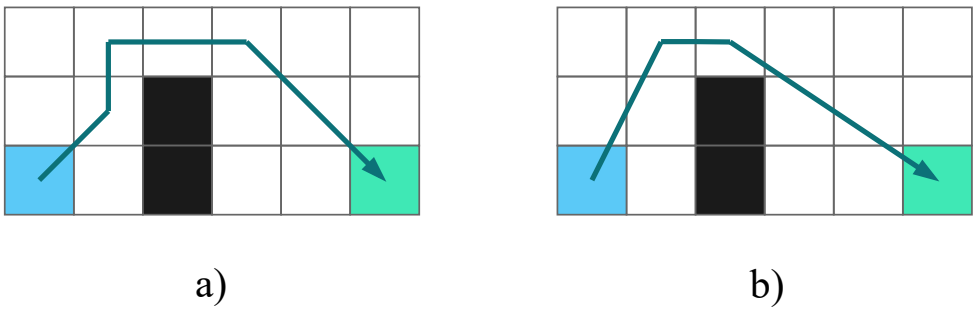
Algoritmus HPA* PS je založený na hierarchickém vyhledávání pomocí algoritmu HPA*. Rozdíl je v tzv. post-processingu neboli dodatečném upravování nalezené cesty. Algoritmus HPA* hledá výslednou cestu v prostředí pravidelných mřížkových map viz kap. 2.2.1. V těchto mřížkách je však umožněn pohyb v nanejvýše 8 směrech (2 horizontální, 2 vertikální, 4 diagonální). To způsobuje jisté omezení a nalezená

cesta tak obvykle není nejkratší možná. Algoritmus HPA* PS využívá techniku *A* Post Smoothing* (A* PS), která upravuje nalezenou cestu pomocí algoritmu A* tak, aby byl umožněn pohyb všemi směry, a tedy pod jakýmkoli úhlem. Pseudokód je uveden v algoritmu č. 2.

Algoritmus 2 *A* PS algoritmus*

```
1:  procedure AStarPostSmoothing(path)
2:      k := 0
3:      smoothedPath := ∅
4:      smoothedPath[k] := path[0]
5:      for each i ∈ [1, ..., (n − 1)] do
6:          if not LineOfSight(smoothedPath[k], path[i + 1]) then
7:              smoothedPath[k] := path[i]
8:              k := k + 1
9:          end if
10:     end for each
11:     smoothedPath[k] := path[n]
12:     return smoothedPath
13: end procedure
```

Vyhlazení cesty spočívá v nahrazení určitých úseků cesty rovnými čarami. Algoritmus A* PS patří mezi tzv. *any-angle* algoritmy [28, 15]. Rozdíl mezi cestou pro pohyb v mřížce a any-angle cestou je uveden na obr. 25.



Obr. 25: (a) Cesta v mřížce (b) Any-angle cesta

Aby bylo docíleno pohybu všemi směry, algoritmus pracuje na principu toho, že zkoumá, zda existuje *vzájemná viditelnost* mezi jednotlivými body cesty. Na počátku algoritmus vybere první uzel nalezené cesty prostřednictvím A* a testuje, zda lze vynechat následující uzel a jít přímo k dalšímu následníkovi. Toto testování je provedeno pomocí algoritmu *Line-of-Sight*. Pokud platí, že existuje vzájemná

viditelnost mezi těmito dvěma uzly, potom je možné prostřední uzel přeskočit a pokračovat přímo na dalšího následníka. U něj se opět testuje, zda nejde přeskočit rovnou k tomuto uzlu a vynechat tak oba předchozí vrcholy. Tento proces se následně opakuje až do doby, kdy vzájemná viditelnost mezi některými dvěma uzly už není splněna. V tomto případě se jako startovací uzel bere poslední, ke kterému byla viditelnost nalezena, a proces začne od tohoto uzlu znovu [12].

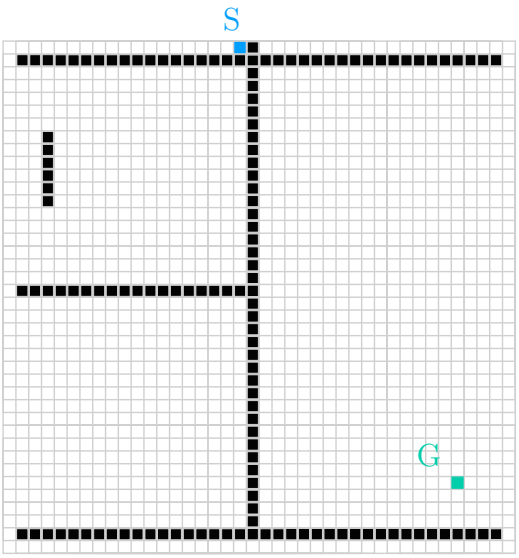
3.5.2 HPA* LEC

Tvorba hierachického prostředí pomocí algoritmu HPA* je poměrně časově náročná. Často však není potřeba prohledávat celé prostředí mapy a hledat vnitřní cesty pro každý shluk v rámci celé hierarchie. Místo toho je možné nalézt vnitřní cesty, tedy intra-hrany ve shlucích, tzv. na vyžádání.

V návaznosti na rozšíření z článku [22] byl navržen hierarchický přístup HPA* LEC, který nevytváří kompletní hierarchické prostředí na počátku, ale sestavuje si ho až v samém průběhu vyhledávání. Jednotlivé oblasti jsou poté buď prohledány nebo neprohledány v závislosti na tom, kterým směrem prohledávání postupuje. Algoritmus tak omezí vytváření hierarchie v místech, které nejsou nezbytné k nalezení výsledné cesty.

Navržený algoritmus funguje tak, že si v rámci inicializačního hierarchického zpracování rozdělí prostředí na shluky pro jednotlivé abstraktní levely a vytvoří spojení mezi nimi, tedy jejich inter-hrany. Intra-hrany uvnitř jednotlivých shluků algoritmus nevytváří. Jakmile je spuštěno vyhledávání, do předpřipraveného hierarchického prostředí je vložen startovací a cílový uzel. Vložení je stejné jako u algoritmu HPA*. Uzly se tedy připojí ke všem místům přechodu, které v daném shluku existují. Následně začne samotné hledání pomocí algoritmu A*. U každého expandovaného uzlu algoritmus zkoumá, zda již byly definovány vnitřní hrany shluku, ve kterém se nachází. Pokud ano, algoritmus pokračuje dále. Pokud ne, algoritmus vytvoří vnitřní hrany všech shluků, ve kterých se pro jednotlivé levely nachází. Informaci o již prohledaných shlucích si ukládá do mezipaměti a tyto shluky již v budoucnu znovu nezpracovává. Proces postupného prohledávání jednotlivých shluků se poté opakuje až do doby, dokud není nalezena výsledná cesta.

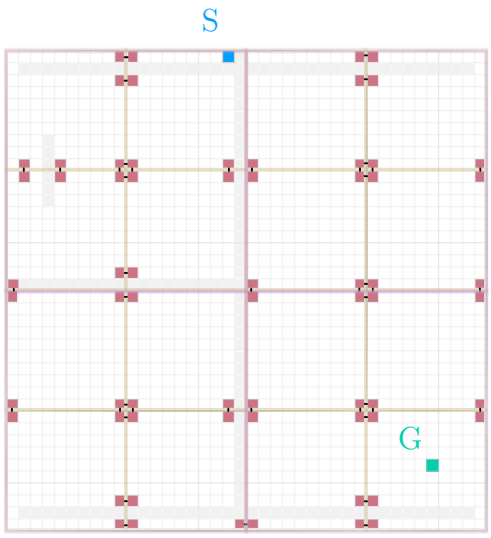
Pro demonstraci algoritmu je využita mapa z ukázky pro algoritmus HPA*. Startovací a cílový bod byly umístěny tak, aby lépe ilustrovaly výhody algoritmu HPA* LEC. Počáteční rozložení mapy je uvedeno na obr. 26.



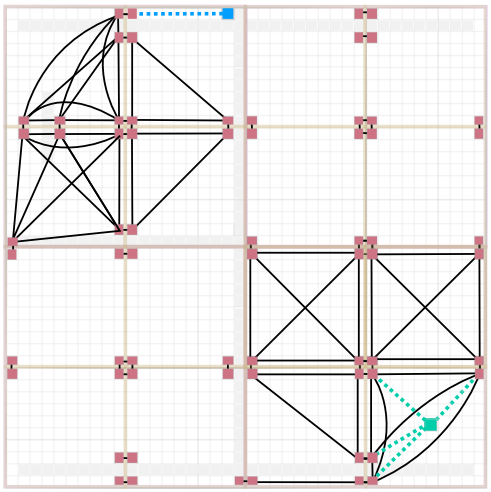
Obr. 26: Výchozí mapa pro algoritmus HPA* LEC

Na obr. 27 je následně zobrazen proces tvorby hierarchického prostředí pomocí algoritmu HPA* LEC. Vidíme, že algoritmus nejprve v kroku 1 definuje všechny shluky pro jednotlivé levely hierarchie a následně mezi nimi vytvoří inter-hrany. V kroku 2 probíhá přípojení startovacího a cílového bodu k hranici shluků na levelu 1, ve kterých se každý z bodů nachází. Připojení k hranici představuje nalezení cesty mezi daným bodem a všemi body přechodu odpovídajícího shluku. V kroku 3 se tento proces opakuje pro shluk na vyšším levelu abstrakce, tedy na levelu 2. Pokud by měla hierarchie více abstraktních levelů, proces by se pro každý level zopakoval. V kroku 4 a 5 je hierarchie rozšířena v závislosti na tom, jak postupovalo vyhledávání. Vidíme, že došlo k nalezení vnitřních cest pro shluky v levé spodní části mapy.

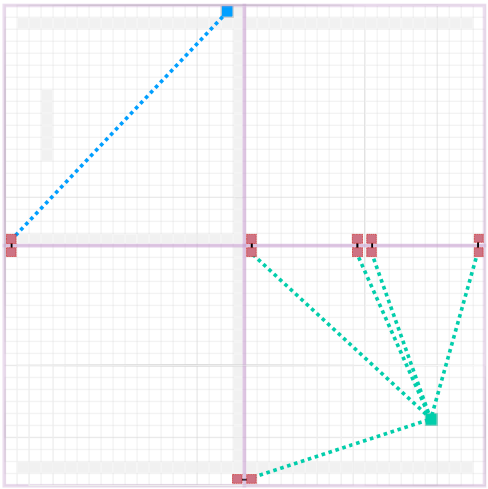
Hierarchická reprezentace prostředí na jednotlivých abstraktních levelech zobrazená v krocích 4 a 5 je již pro dané vyhledávání konečná. Vidíme, že nebyla vytvořena celá grafová reprezentace výchozího prostředí, ale pouze její část. To vedlo ke snížení nároků na tvorbu hierarchie, která představuje časově nejnáročnější část hierarchického vyhledávání. Algoritmus tak našel výsledné řešení rychleji než původní HPA*. V tom tkví efektivita použití daného algoritmu.



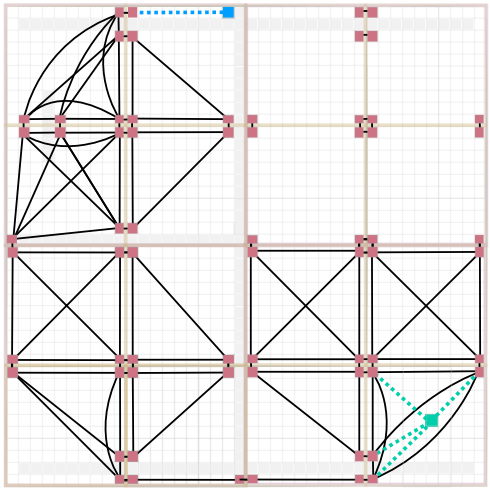
Krok 1



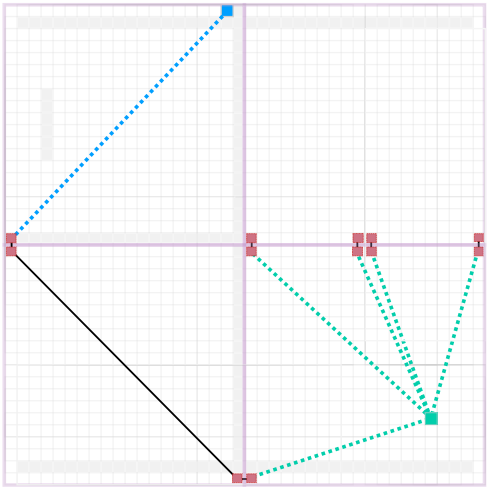
Krok 2



Krok 3



Krok 4



Krok 5

Obr. 27: Tvorba grafové reprezentace pro algoritmus HPA* LEC

4 POPIS APLIKACE

V následující kapitole bude popsána aplikace *HierarchicalPathfinding*, která byla vytvořena v rámci praktické části diplomové práce. Pro vytvoření aplikace byl použit programovací jazyk C# a vývojové prostředí *Visual Studio 2022*.

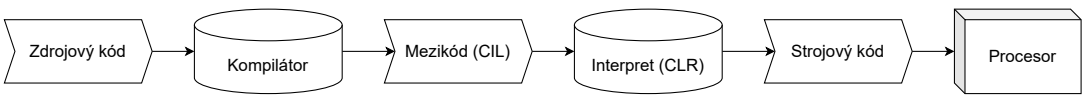
4.1 Použité technologie

Programovací jazyk C#

Jazyk C# představuje moderní vysokoúrovňový objektově orientovaný jazyk vyvinutý firmou *Microsoft*. Byl představen spolu s vývojovým prostředím *.NET*. Své kořeny má v jazyce *C*, *C++* a *Java*. Vyznačuje se vlastnostmi jako například zajištění typové bezpečnosti, kontrola hranic polí, jednoduchá dědičnost nebo zpracování chyb pomocí výjimek. Jazyk je silně typovaný a má automatické uvolňování paměti, které zajišťuje tzv. *garbage collector*. Je vhodný pro vývoj softwarových komponent pro nasazení v distribuovaných prostředích [5, 10, 34].

Programy v jazyce C# jsou založeny na softwarové technologii .NET. Ta obsahuje celou řadu knihoven, které poskytují širokou škálu užitečných funkcí, a to například pro práci s konzolí, databázemi nebo formulářovými prvky. Existuje několik implementací .NET. Mezi ně patří *.NET Framework*, *.NET Core*, *Mono* a *.NET Standard* [5, 10].

Struktura jazyka C# využívá tzv. *kompilátor* a *interpret* viz obr. 28. Zdrojový kód je nejprve pomocí kompilátoru přeložen do tzv. mezikódu s názvem *Common Intermediate Language* (CIL). Jedná se v podstatě o strojový kód s instrukční sadou podporující objektové programování. Typicky jsou to soubory s příponou *.dll*. Tento kód je následně interpretován pomocí modulu *Common Language Runtime* (CLR), který poskytuje výsledný strojový kód pro procesor počítače. V rámci rozhraní .NET představuje modul CRL virtuální stroj neboli interpret [5, 10].



Obr. 28: Struktura programovacího jazyka C# (podle [10])

Windows Presentation Foundation

Pro návrh grafického uživatelského rozhraní (GUI) bylo použito rozhraní *Windows Presentation Foundation* (WPF). Jedná se o vektorový vykreslovací engine navržený tak, aby využíval výhod moderního grafického hardwaru. Technologie je součástí

.NET a používá jazyk *Extensible Application Markup Language* (XAML). WPF poskytuje komplexní sadu funkcí pro vývoj aplikací, které zahrnují například ovládací prvky, datové vazby, rozvržení, 2D a 3D grafiku, typografii nebo zabezpečení [40].

Stylet
Knihovna *Stylet* poskytuje výkonný rámec *Model-View-ViewModel* (MVVM) architektury. Jeho záměrem je snížit složitost při práci s MVVM frameworky. Ty jsou základem i pro rozhraní WPF. Stylet umožňuje uživateli psát udržovatelný a rozšiřitelný kód způsobem, který lze snadno testovat. Pracuje na principu *ViewModel-first*, který je protikladem klasické struktury View-first. V rámci ViewModel-first přístupu je nejprve vytvořena instance ViewModelů, zatímco Views jsou poté připojeny automaticky. V rámci knihovny lze využít funkcionality pro řízení akcí a událostí (Actions), práci s ViewModelem (Screen a Conductor), zachycení uživatelské interakce a zasílání zpráv mezi ViewModely (The EventAggregator), správu oken (The WindowManager), spouštění validací a hlášení výsledků (Validation) a možnost uvolňování pevné vazby mezi objekty pomocí rychlého IoC kontejneru (StyletIoC) [37]

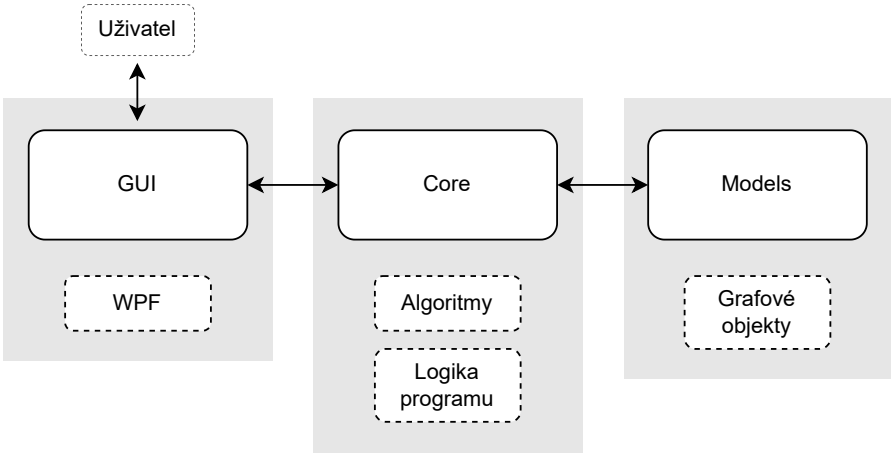
Priority Queue
Algoritmus A* používá v rámci vyhledávání prioritní frontu. V projektu je využita *SimplePriorityQueue* z knihovny *High Speed Priority Queue* [19]. Ta poskytuje veškeré požadované vlastnosti prioritní fronty. Vyznačuje se také například stabilitou, bezpečnostními kontrolami nebo dynamickou změnou velikosti.

Moving AI Benchmarks
Aplikace umožňuje zpracovávat vstupní soubory mapového formátu *.map*, který zavedl Nathan Sturtevant v článku *Benchmarks for Grid-Based Pathfinding* [4]. Formát těchto map obsahuje hlavičku o čtyřech řádcích obsahující informace o typu a velikosti mapy. Následuje definice dat prostředí, které je reprezentováno sadou znaků. Ty definují jednotlivé prvky prostředí, které následně představují buď překážku nebo prostupnou oblast mapy. Tento typ map byl použit v řadě počítačových videoher, kterými jsou například *Dragon Age*, *Warcraft III* nebo *Starcraft*. K dispozici jsou zde například i mapy měst (London, Paris) nebo různá bludiště. Všechny tyto mapy jsou volně dostupné online viz [30].

4.2 Struktura programu

Obrázek č. 29 popisuje strukturu aplikace *HierarchicalPathfinding*. Uživatel přistupuje k aplikaci pomocí grafického uživatelského rozhraní. To je umístěno v samostatném projektu s názvem **GUI**. Grafické rozhraní obsahuje prvky MVVM architektury a je implementováno prostřednictvím WPF. Programová logika a třídy algoritmů jsou uloženy v projektu **Core**. Zde je prováděno samotné plánování cesty. V pro-

jektu **Models** jsou uloženy třídy grafových objektů. Ty tvoří základ pro grafickou reprezentaci vstupního prostředí mapy. Následně jsou jeho prvky využity pro tvorbu abstraktního grafu, na kterém je prováděno vyhledávání.



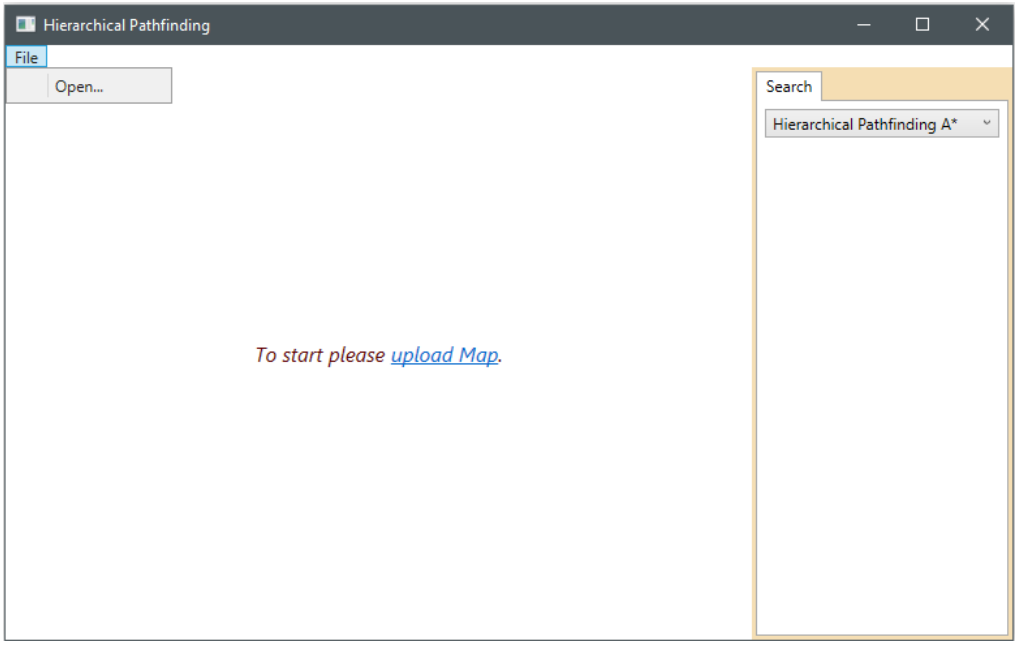
Obr. 29: Struktura programu

4.3 Uživatelské rozhraní

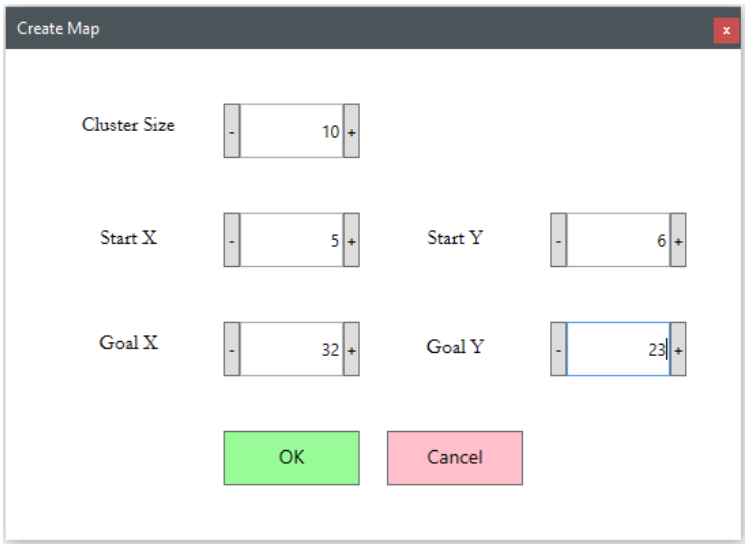
Uživatelské rozhraní bylo navrženo za účelem snadné interakce uživatele s aplikací. Na obr. 30 je uvedeno startovací okno aplikace. To je rozděleno do dvou hlavních částí. Oblast vlevo poskytuje prostor pro zobrazení mapy prostředí. Vpravo je umístěno hlavní menu aplikace. Startovací okno umožňuje uživateli nahrát vstupní soubor ve formátu *.map*. Soubor je možné nahrát buď pomocí odkazu uprostřed okna, nebo pomocí rozbalovacího seznamu **File** a příkazu **Open..** v levé horní části aplikace. Prostřednictvím tohoto rozbalovacího seznamu lze následně přidat více vstupních map, které budou otevřeny v jednotlivých záložkách.

Po nahrání vstupního souboru je možné nastavit vstupní parametry pro dané vyhledávání. Vstupní parametry se nastavují prostřednictvím tlačítka **Set Parameters**. Po jeho stlačení se objeví dialogové okno viz obr. 31. Uživatel zde zadá požadovanou velikost shluků (Cluster Size). Tato hodnota odpovídá velikosti shluků na úrovni levelu 1. Velikost shluků na dalších úrovních je definována uvnitř programu. Následně si uživatel zvolí souřadnice startovacího a cílového bodu.

Po nastavení parametrů a stisknutí tlačítka **OK** je možné spustit vyhledávání programu pomocí zvoleného algoritmu. To lze provést stisknutím tlačítka **Run Search**, které se nově zobrazí v oblasti menu. Požadovaný algoritmus si uživatel volí z nabídky rozbalovacího seznamu v horní části aplikačního menu viz obr. 32.

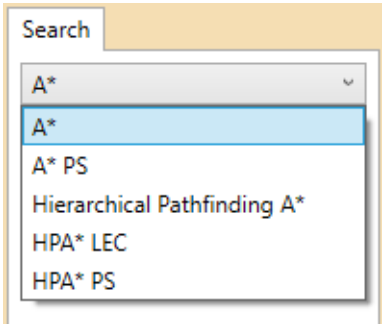


Obr. 30: Startovací okno aplikace



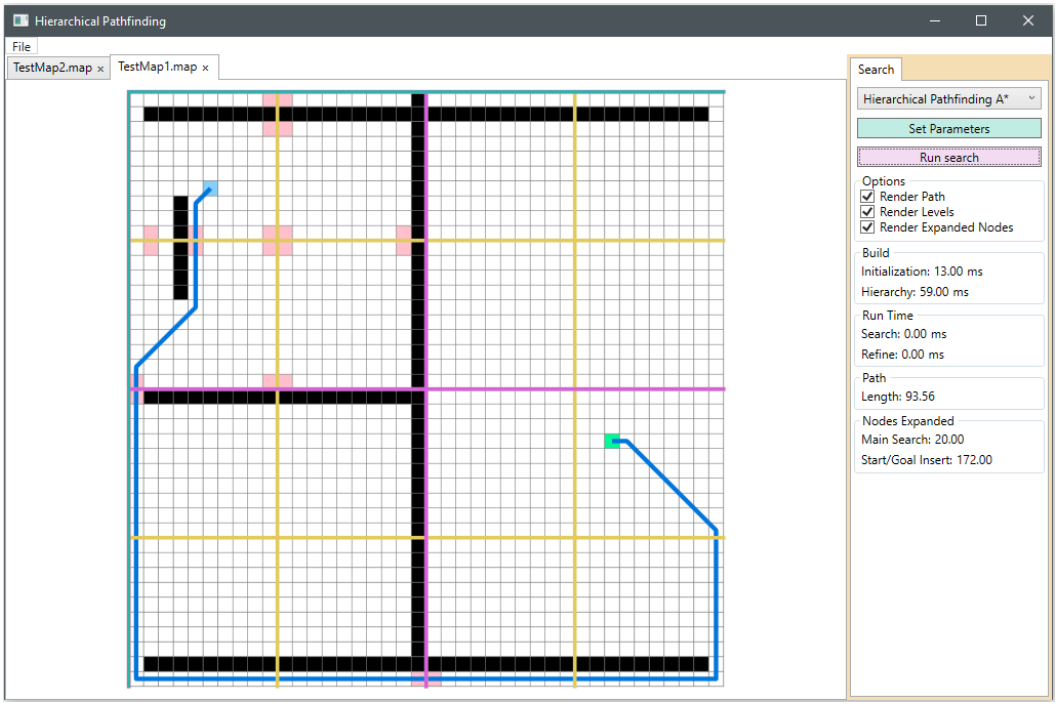
Obr. 31: Nastavování vstupních parametrů

Po skončení vyhledávání se v prostředí mapy vykreslí nalezená cesta spolu s expandovanými uzly viz obr. 33. Expandované uzly jsou v mapě reprezentovány červenými poli. Pokud se jedná o hierarchické vyhledávání, jsou v mapě vykresleny i hranice shluků pro jednotlivé abstraktní levely. Všechna tato vykreslení je možné deaktivovat v sekci **Options**, která se nově objeví v oblasti menu. Zde jsou uvedeny také výsledky vyhledávání. V sekci **Build** je zobrazena doba inicializace (Initialization) a doba tvorby hierarchického prostředí (Hierarchy). Doba inicializace za-



Obr. 32: Rozbalovací seznam pro výběr algoritmů

hrnuje zpracování vstupního prostředí do grafové struktury. Tvorba hierarchického prostředí představuje vytvoření jednotlivých levelů abstrakce a připojení startovacího a cílového bodu. Sekce **Run Time** poté udává výslednou dobu vyhledávání (Search) a dobu potřebnou pro zpřesnění nalezené cesty (Refine). **Path** obsahuje informaci o celkové délce nalezené cesty (Length). V poslední sekci s názvem **Nodes Expanded** lze vyčíst počet axpandovaných uzlů pro hlavní vyhledávání (Main Search) a počet uzlů, které byly expandovány v procesu připojení startovací a cílové pozice do jednotlivých vrstev abstraktního grafu (Start/Goal Insert).

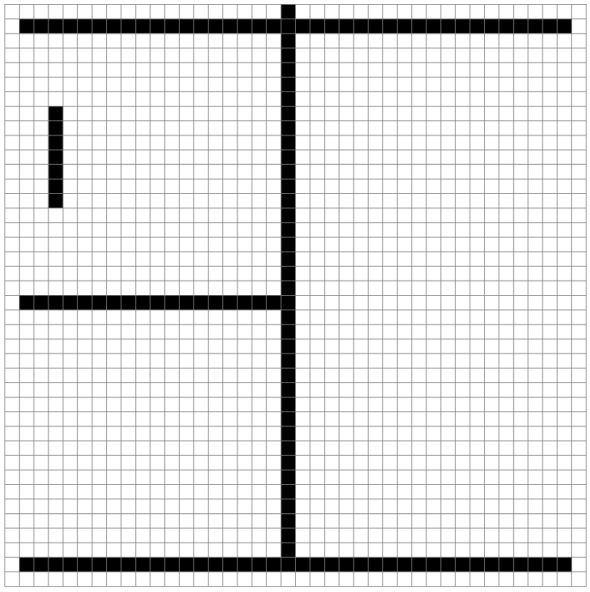


Obr. 33: Uživatelské rozhraní aplikace

5 EXPERIMENTÁLNÍ ČÁST

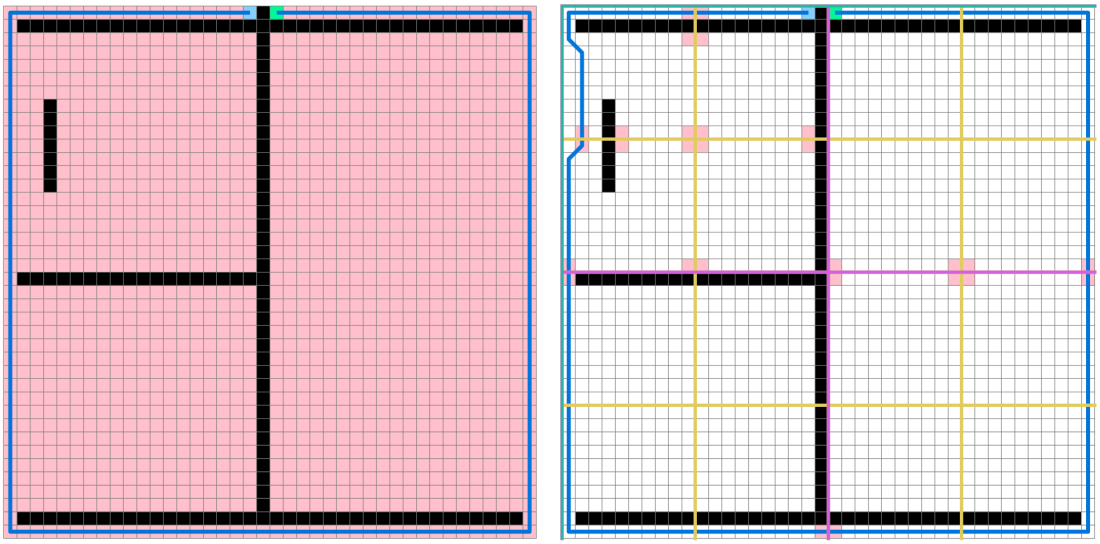
V následující kapitole budou představeny jednotlivé experimenty a jejich výsledky. Cílem bylo porovnat efektivitu vybraných algoritmů. Ve vytvořeném simulačním prostředí bylo provedeno celkem 7 experimentů na několika různých mapách. Použité mapy jsou buď vytvořené za účelem testování, nebo byly použity již existující mapy ze sady MovingAI [4]. Tyto mapy jsou blíže popsány v kapitole 4.1. Testování běhů algoritmů bylo v rámci jednotlivých experimentů provedeno opakovaně, aby byly dosažené výsledky statisticky vypovídající. Každý test je tak výsledkem 40-ti opakovaných spuštění.

Experiment č. 1
K tomuto experimentu byl vybrán typ mapy ze článku *Near Optimal Hierarchical Path-Finding* [7]. Jedná se o mapu velikosti 40x40 uvedenou na obr. 34. Na této mapě bylo provedeno srovnání tradičního způsobu vyhledávání pomocí algoritmu A* a hierarchického přístupu hledání cesty prostřednictvím algoritmu HPA*.



Obr. 34: Testovací mapa č. 1

Výsledná cesta a počet expandovaných uzlů pro oba algoritmy jsou zobrazeny na obr. 35. Všimněme si zde samotného charakteru nalezené cesty. Lze vidět, že pro algoritmus HPA* byla nalezena suboptimální cesta. Tedy cesta, která není nejkratší. To je způsobeno přesně definovanými místy přechodu, kterými algoritmus musí procházet. Tím je definován suboptimální charakter výsledných cest. Z toho důvodu je také výsledná cesta pro HPA* o něco delší.



Obr. 35: Porovnání algoritmů A* a HPA*

V tab. 3 jsou uvedeny výsledné hodnoty vyhledávání. Můžeme vidět, že je tato mapa pro algoritmus A* značně náročná, neboť pro nalezení konečné nejkratší cesty bylo nutné expandovat všechny uzly mapy. V případě hierarchického přístupu byl počet prohledaných uzlů výrazně menší. Je nutné podotknout, že v rámci hierarchického přístupu je nutné započítat také počet expandovaných uzlů potřebných pro připojení startovacího a cílového bodu. Celkový počet prohledaných uzlů je pak u HPA* zhruba 30x menší.

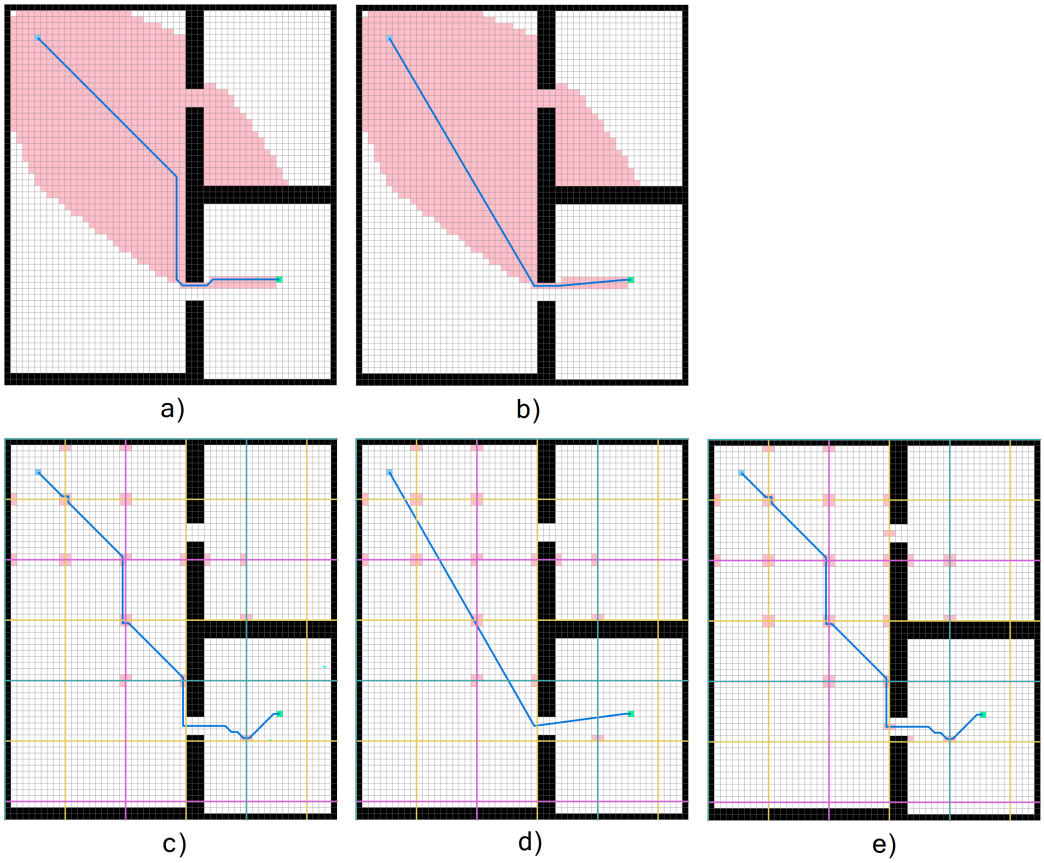
Tab. 3: Výsledky experimentu č. 1

	Tvorba hierarchie	Vyhledávání a zpřesnění		Délka cesty	Vyhledávání	Připojení startu a cíle
	[ms]	[ms]	[ms]	[-]	Expandované uzly	
A*	0.00	4.00	0.00	154.00	1462	0
HPA*	62.00	0.00	0.00	154.83	28	19

Co se týče celkové doby běhu, je algoritmus A* výrazně rychlejší. To je způsobeno časově poměrně náročnou tvorbou hierarchického prostředí na počátku vyhledávání. Tvorba hierarchie je však vždy provedena pouze jednou a další běh algoritmu zahrnuje již jen samotné vyhledávání na nejvyšší úrovni abstrakce. Ta obvykle obsahuje jen malé množství uzlů. Z tohoto důvodu je samotné hledání pomocí HPA* obecně výrazně rychlejší než u tradičního algoritmu A*. Čas hledání HPA* se v uvedeném případě pro malou velikost vstupní mapy téměř blíží nule.

Experiment č. 2

Tento experiment byl proveden za účelem porovnání charakteru výsledných cest pro jednotlivé implementované algoritmy. Experiment byl prováděn na mapě menší velikosti, aby byl charakter cesty dobře viditelný. Konkrétně se jedná o mapu ze sady MovingAI s názvem **isound1.map** a velikostí 63x55. Velikost shluků byla volena 10. Mapa je uvedena na obr. 36 spolu s vykreslenými cestami pro jednotlivé algoritmy.



Obr. 36: (a) A* (b) A* PS (c) HPA* (d) HPA* PS (e) HPA* LEC

Na obrázku jsou viditelné odlišnosti daných cest a počty expandovaných uzlů pro jednotlivá vyhledávání. V horní části je uveden tradiční algoritmus A* spolu s any-angle algoritmem A* PS. Použití algoritmu A* PS vedlo k nalezení kratší a přímočařejší cesty k cíli. Červeně jsou v obrázcích zaznačeny expandované uzly. Počet těchto uzlů je pro oba zmíněné algoritmy stejný. V dolní části jsou zobrazeny algoritmy využívající hierarchické přístupy. První z nich c) popisuje algoritmus HPA*. Jsou zde vykresleny jednotlivé levely hierarchie a expandované uzly na nejvyšší úrovni, kde je prováděno samotné vyhledávání. Vidíme, že algoritmus opět našel suboptimální cestu. Na obrázku d) je vidět any-angle algoritmus HPA* PS. Ten expandoval stejné uzly jako algoritmus HPA*, avšak díky vyhlazení cesty poskytuje přímočařejší a kratší cestu k cíli (podobně jako A* PS). Na posledním

obrázku je zachycen algoritmus HPA* LEC. Jeho cesta je v tomto případě stejná jako u algoritmu HPA*, liší se však v počtu expandovaných uzlů.

Informace o nalezených cestách byly zaznamenány v tab. 4. Z hlediska počtu expandovaných uzlů lze pozorovat výraznou redukci při použití hierarchických metod. Jako nejefektivnější se jevil algoritmus HPA*. HPA* LEC expandoval celkově o něco více uzlů než HPA*, neboť na rozdíl od něj nemá k dispozici celé hierarchické prostředí a pohybuje se tak o něco méně přímočaře k cíli.

Vzhledem k délce nalezené cesty se jako nejefektivnější jevil algoritmus A* PS. Druhou nejkratší cestu měl any-angle algoritmus HPA* PS. Jeho výsledná délka byla kratší než pro tradiční A*. Vzhledem k tomu, že hierarchické metody vrací suboptimální cesty, které na rozdíl od tradičního A* nemusí být nejkratší, je toto poměrně zajímavý výsledek, který může vést ku prospěchu použití hierarchických metod.

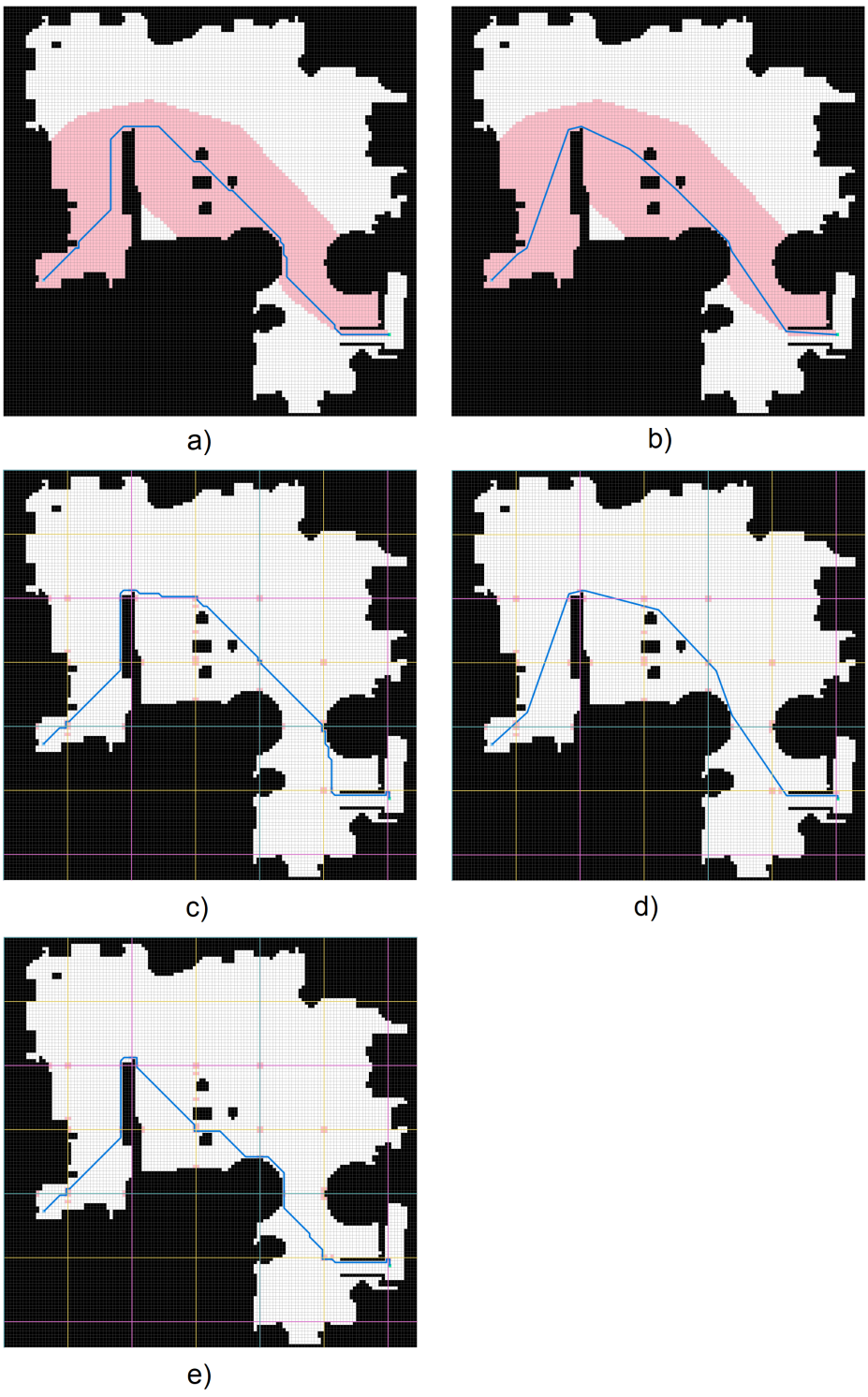
Tab. 4: Výsledky experimentu č. 2

	Délka cesty	Vyhledávání	Připojení startu a cíle
	[-]	Expandované uzly	
A*	67.36	1204	0
A* PS	63.97	1204	0
HPA*	71.60	42	46
HPA* PS	64.82	42	46
HPA* LEC	71.60	61	46

Experiment č. 3

Experiment byl proveden s cílem provést srovnání běhů jednotlivých algoritmů. K experimentu byla vybrána testovací mapa **den901d.map** o velikosti 128x129 ze sady MovingAI. Velikost shluků je 20. Mapu lze vidět na obr. 37 spolu s výslednými nalezenými cestami pro jednotlivé algoritmy. V tab. 9 jsou uvedeny výsledky běhu jednotlivých algoritmů. Doba inicializace, tedy tvorba grafové reprezentace výchozího prostředí, zde není uvedena, neboť je pro všechny algoritmy stejná. Vzhledem k poměrně malé velikosti mapy a relativně nízké hustotě překážek jsou časy pro vyhledávání algoritmů nízké. Zejména u hierarchických přístupů se tyto časy blíží nule.

Pokud porovnáme výsledky pouze jednoho běhu u jednotlivých algoritmů, lze říci, že pro danou mapu je z časového hlediska výhodné použít samotný algoritmus A*. Ten byl pro jedno spuštění obecně rychlejší než algoritmus HPA*. Výhody hierarchického přístupu by se projevily zejména ve větších mapách a při opakovaném vyhledávání, kdy se doba pro tvorbu hierarchie omezí pouze na vložení startovní a



Obr. 37: (a) A* (b) A* PS (c) HPA* (d) HPA* PS (e) HPA* LEC

cílové pozice. Zbytek hierarchického prostředí zůstane uložený z prvního vyhledávání.

Tab. 5: Výsledky experimentu č. 3

	Tvorba hierarchie	Vyhledávání a zpřesnění		Délka cesty	Vyhledávání	Připojení startu a cíle
	[ms]	[ms]	[ms]	[-]	Expandované uzly	
A*	0.00	11.00	0.00	175.31	3319	0
A* PS	0.00	11.00	0.00	167.14	3319	0
HPA*	293.00	0.00	1.00	185.51	78	60
HPA* PS	313.00	0.00	1.00	172.67	78	60
HPA* LEC	111.00	144.00	0.00	184.34	80	60

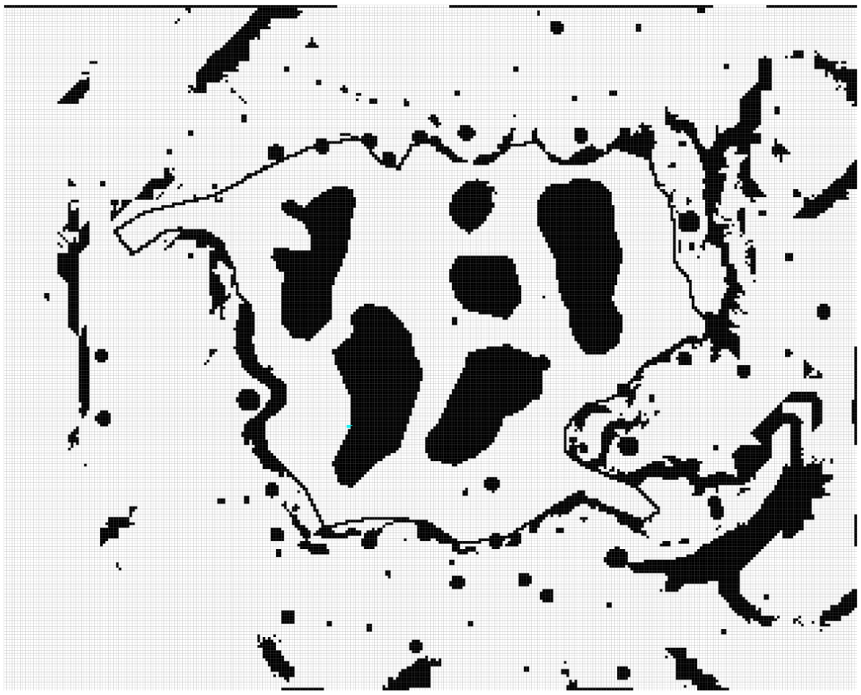
Zaměříme-li se na počet expandovaných uzlů, lze pozorovat, že hierarchické metody obecně expandují během samotného vyhledávání výrazně menší počet uzlů. Z tohoto hlediska se tedy hierarchický přístup ukázal jako efektivnější. Nejlepších výsledků dosahoval opět algoritmus HPA*. Charakteristika délek nalezených cest je obdobná jako v případě experimentu č. 2. Lze zde však pozorovat, že na rozdíl od mapy použité v předchozím experimentu (obr. 36) HPA* a HPA* LEC nenašly stejnou cestu. Nalezená cesta pro HPA* LEC je v tomto případě o něco kratší.

Experiment č. 4

V tomto experimentu byl testován vliv velikosti shluků na tvorbu hierarchického prostředí a následně i na samotné hierarchické vyhledávání. Velikost shluků je zadávána uživatelem na počátku vyhledávání. Tato velikost odpovídá rozměrům shluků na úrovni levelu 1. Ostatní levely jsou dopočítány v rámci algoritmu automaticky. Jako testovací mapa byla vybrána **brc503d.map** o velikosti 257x320 viz obr. 38.

Na mapě bylo postupně spuštěno hierarchické vyhledávání pomocí algoritmu HPA* pro velikosti shluků 20, 50, 70, 100 a 150. Zajímá nás především doba tvorby hierarchického prostředí pro první běh algoritmu a čas pro připojení startu a cíle v dalších spuštěních (aktualizace hierarchie). Výsledky pro jednotlivé případy jsou uvedeny v tab. 6.

Ze získaných výsledků lze pozorovat, že se zvyšující se velikosti počátečních shluků obecně klesá doba potřebná pro tvorbu hierarchie. Výjimkou byl shluk o velikosti 100, který zřejmě nebyl pro daný typ mapy příliš vhodný a tvorba hierarchie zde trvala delší dobu. Čas pro vyhledávání je pro všechny případy srovnatelný. Můžeme si však všimnout, že s většími shluky je potřeba více času na zpřesnění cesty. Aktualizace hierarchie představuje opětovné spuštění algoritmu na již vytvořeném hierarchickém prostředí. Zde můžeme vidět vzestupnou tendenci s rostoucí velikostí shluků. Časové náklady na připojení startu a cíle totiž u větších počátečních shluků



Obr. 38: Testovací mapa č. 4

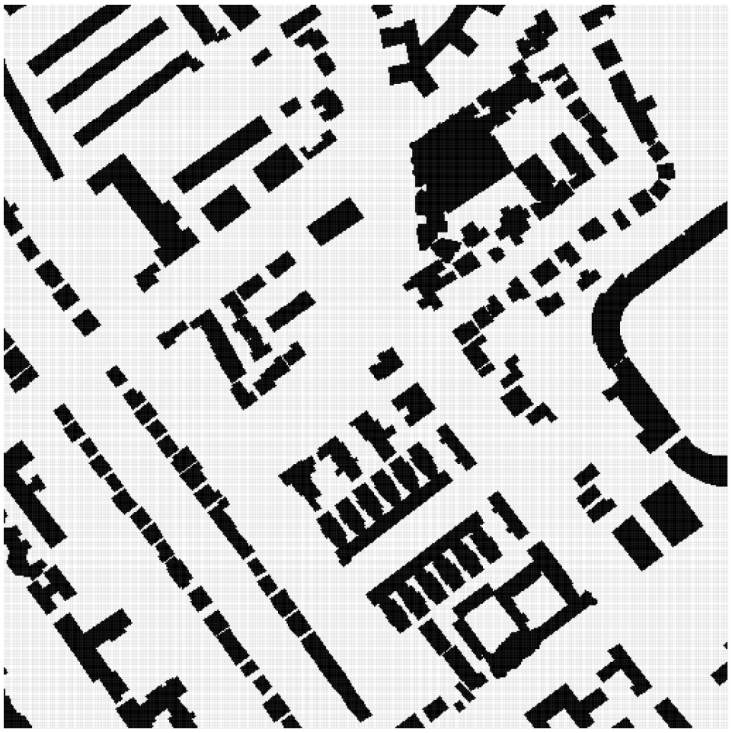
Tab. 6: Vliv velikosti shluků na hierarchické vyhledávání

Velikost shluků	Tvorba hierarchie	Vyhledávání a zpřesnění		Aktualizace hierarchie	Vyhledávání	Připojení startu a cíle
[-]	[ms]	[ms]	[ms]	[-]	Expandované uzly	
20	210795	1.00	0.00	4.00	235	317
50	161175	1.00	0.00	31.00	98	1277
70	115833	1.00	1.00	168.00	58	13628
100	185278	0.00	1.00	458.00	44	35236
150	102513	0.00	2.00	184.34	20	44548

rostou, neboť prohledáváme větší část základních uzlů mapy a méně levelů hierarchie. Zaměříme-li se na počet expandovaných uzlů během samotného vyhledávání, vidíme, že jejich počet během hierarchického vyhledávání klesá s rostoucí velikostí shluků. Klesání není ovšem tak výrazné jako související vzrůst nákladů na počet prohledaných uzlů pro přípojení startu a cíle. Ty velmi výrazně rostou se zvyšující se velikostí shluků.

Experiment č. 5

Tento experiment byl proveden za účelem srovnání tradičního a hierarchického přístupu k plánování cesty ve větších mapách. Pro porovnání byly vybrány tradiční algoritmus A* a hierarchický algoritmus HPA*. K provedení experimentu byly použity mapy **London_2_256.map**, **London_2_512.map** a **London_2_1024.map** s velikostmi 256x256, 512x512 a 1024x1024. Charakter mapy je pro všechny velikosti stejný viz obr. 39, liší se pouze rozlišení jednotlivých map.



Obr. 39: Testovací mapa č. 5

Výsledky běhů jsou uvedeny v tab. 7. Srovnáme-li první běhy algoritmů HPA* a A*, nejkratší doba vyhledávání je ve všech případech u samotného algoritmu A*. To je zapříčiněno opět tvorbou hierarchie u HPA*. Podíváme-li se pouze na samotný běh obou algoritmů, vidíme, že doba vyhledávání je u HPA* výrazně nižší než u A*.

V případě největší mapy je doba nutná pro hierarchické vyhledání a zpřesnění až 240x nižší. Doba vyhledávání HPA* spolu se zpřesněním nalezené cesty zde zabere 16 ms, zatímco A* algoritmus provádí hledání téměř 4000 ms. Je však důležité do tohoto součtu zahrnout také dobu nutnou pro vložení startu a cíle do jednotlivých hierarchických levelů. V takovém případě není rozdíl ve vyhledávání tak obrovský, avšak stále pozorujeme u HPA* výrazně lepší výsledky. Celkový čas pro porovnání je pak u algoritmu A* 3829 ms a u algoritmu HPA* 1179 ms. Čas potřebný pro vyhledávání se tak snížil o více než polovinu. Obecně můžeme pozorovat, že s rostoucí

velikostí mapy rostou výhody použití algoritmu HPA*. A to jak z časového hlediska, tak především z hlediska výpočetní náročnosti, kdy v rozsáhlých mapách algoritmus A* expanduje velké množství uzlů vůči HPA*.

Tab. 7: Výsledky experimentu č. 5

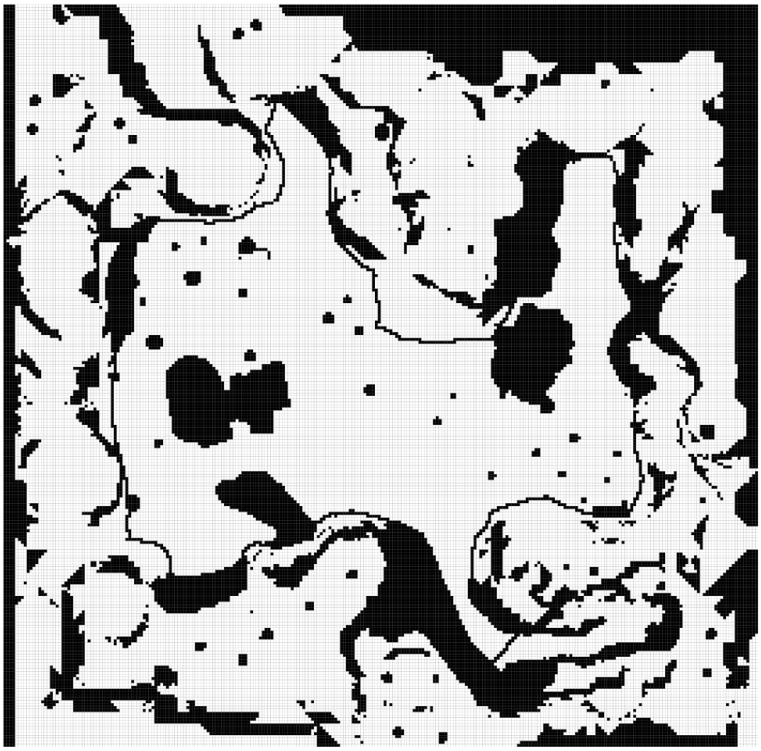
	Tvorba hierarchie	Vyhledávání a zpřesnění		Vyhledávání	Připojení startu a cíle
	[ms]	[ms]	[ms]	Expandované uzly	
			Mapa velikosti 256x256		
A*	0.00	93.00	0.00	22715	0
HPA* (první běh)	28049.00	1.00	0.00	133	4601
HPA* (druhý běh)	51.00	1.00	0.00	133	4601
			Mapa velikosti 512x512		
A*	0.00	385.00	0.00	102505	0
HPA* (první běh)	240117.00	2.00	4.00	138	7809
HPA* (druhý běh)	252.00	2.00	4.00	138	7809
			Mapa velikosti 1024x1024		
A*	0.00	3829.00	0.00	441590	0
HPA* (první běh)	2711913.00	3.00	13.00	146	29905
HPA* (druhý běh)	1163.00	3.00	13.00	146	29905

Zaměříme-li se na počet expandovaných uzlů, vidíme u algoritmu HPA* velmi výrazné snížení. Konkrétně pro největší mapu je podíl expandovaných uzlů pro A* 441 590 a pro vyhledávání HPA* pouze 146. Nejvíce uzlů je v rámci hierarchického vyhledávání prozkoumáno opět z důvodu vložení startovní a cílové pozice. Celkový poměr pro hledání je potom 441 590 (A*) ku 30 051 (HPA*). To je znatelná redukce o téměř 94% uzlů nutných k expandování.

Experiment č. 6

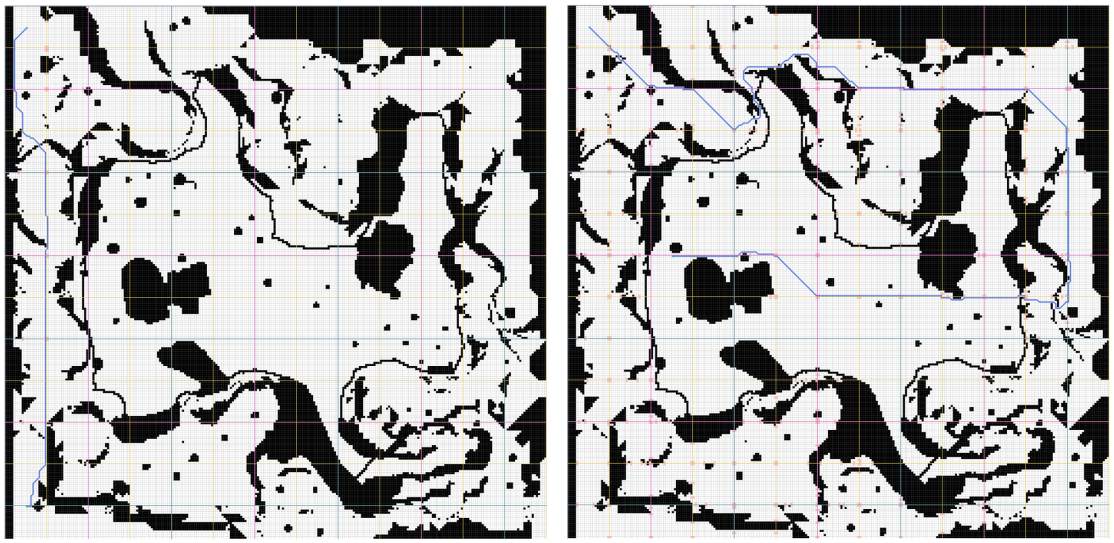
Cílem tohoto experimentu je provést srovnani algoritmů HPA* a HPA* LEC. Pro experiment byla vybrána mapa **brc504d.map** o velikosti 256x260 viz obr. 40. Velikost shluků je volena 20.

V mapě byla startovací a cílová pozice umístěna odlišně pro dva různé scénáře vyhledávání. První pozice startu a cíle byly vloženy k levému okraji mapy (scénář 1). Toto rozložení mělo být příznivější pro HPA* LEC, neboť pro nalezení cesty by měl prohledat pouze menší část mapy. Druhé rozložení startu a cíle bylo naopak voleno



Obr. 40: Testovací mapa č. 6

tak, aby vyžadovalo prohledání velkého množství shluků celé mapy (scénář 2). Jednotlivé scénáře jsou pro upřesnění uvedeny na obr. 41. Vyobrazená cesta zde byla nalezena pomocí algoritmu HPA* LEC.



Obr. 41: (a) Scénář č. 1 (b) Scénář č. 2

V tab. 8 vidíme výsledky vyhledávání pro zmíněné dva scénáře. Zaměříme-li se na získané výsledky, vidíme, že oba algoritmy mají určitý čas vymezený pro tvorbu hierarchie. U algoritmu HPA* je zde vytvořeno celé hierarchické prostředí se všemi vnitřními i vnějšími hranami (inter/intra hrany). Co se týče algoritmu HPA* LEC, jsou v rámci tohoto hierarchického zpracování vytvořeny pouze inter-hrany. Důležitý je pro nás celkový čas pro tvorbu hierarchie a samotného vyhledávání a zpřesňování. V obou případech vidíme, že algoritmus HPA* LEC vykazuje lepší výsledky co se týče redukce času nutného k nalezení výsledné cesty. Pro scénář č. 1 je tato redukce výraznější, neboť je potřeba prohledat jen malou oblast mapy. HPA* LEC zde našel trasu téměř 5x rychleji. U komplikovanější trasy výhoda použití HPA* LEC klesá. Algoritmus je však i v tomto případě účinnější a doba nalezení cesty u něj byla 1.5x kratší. Nutno ale poznamenat, že na rozdíl od HPA* nemá obecně po prvním běhu vytvořené celé hierarchické prostředí. V dalším kroku, pokud by trasa zasáhla do oblastí, které zatím nebyly prohledány, by bylo hledání jistě delší než u HPA*. Co se týče celkového počtu prohledaných uzlů, je v obou případech výhodnější algoritmus HPA*, který expandoval méně uzlů.

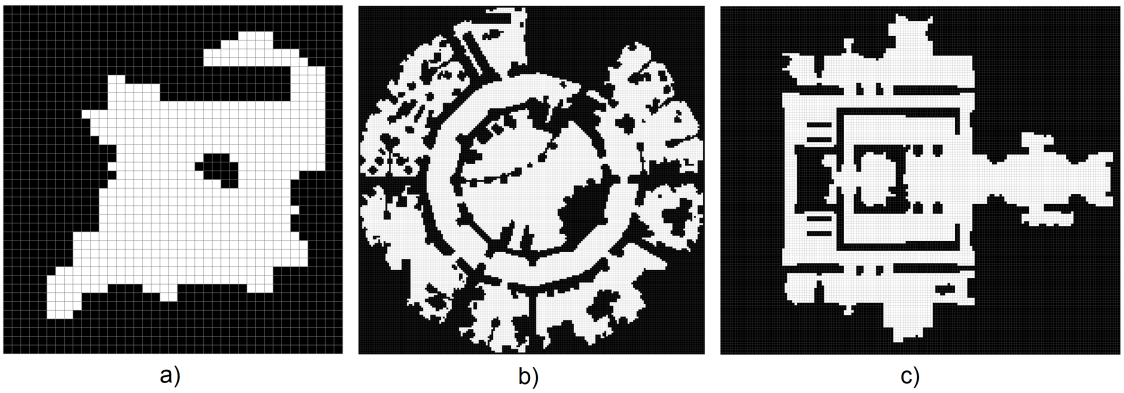
Tab. 8: Výsledky experimentu č. 6

	Tvorba hierarchie	Vyhledávání a zpřesnění		Vyhledávání	Připojení startu a cíle
	[ms]	[ms]	[ms]	Expandované uzly	
			Scénář 1: Jednodušší trasa		
HPA*	148302.00	1.00	0.00	46	172
HPA* LEC	14171.00	15780.00	0.00	91	172
			Scénář 2: Komplikovanější trasa		
HPA*	145939.00	4.00	0.00	509	344
HPA* LEC	13679.00	86242.00	0.00	730	344

Experiment č. 7

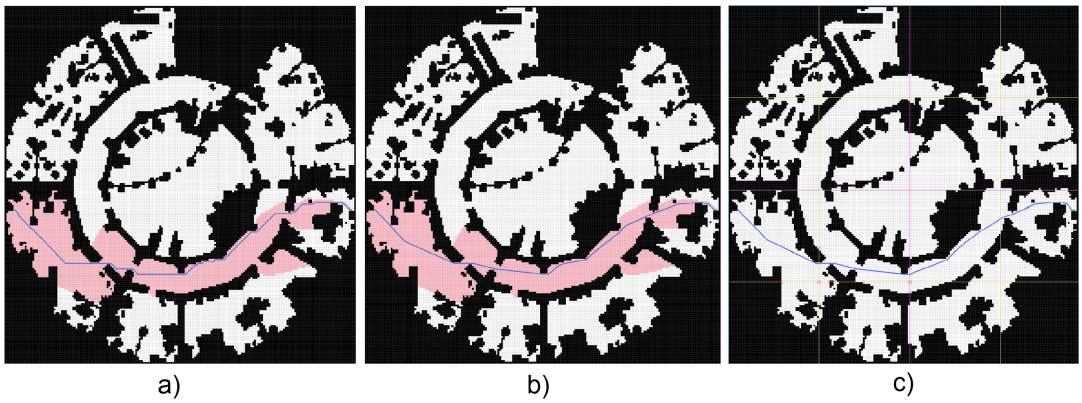
Poslední experiment provádí srovnání algoritmů, které v experimentu č. 2 vrátily nejkratší cesty. Záměrem je porovnat efektivitu jednotlivých algoritmů při testování na různých mapách. Efektivita bude porovnána z hlediska celkové délky cesty, vyhledávacího času a počtu expandovaných uzlů. Pro testování byly vybrány celkem 3 mapy. První je jednoduchá mapa **den202d.map** o velikosti 40x39 s malým množstvím překážek. Druhá mapa **lak504d.map** obsahuje již větší množství překážek a má rozměry 194x193. Třetí mapa s názvem **ht_chantry.map** je mapa méně

výhodná pro A*, neboť obsahuje překážku přibližne ve tvaru U, přes kterou bude provedeno hledání. Mapy jsou zobrazeny na obr. 42.



Obr. 42: (a) Mapa č. 1 (b) Mapa č. 2 (c) Mapa č. 3

Výsledky běhu jednotlivých algoritmů na zmíněných mapách jsou uvedeny v tab. 9. Lze pozorovat, že nejúspěšněji hledá nejkratší cestu algoritmus A* PS. Zajímavý úkaz nastal v případě mapy č. 2, kdy HPA* PS vrací nejkratší cestu ze všech tří algoritmů viz obr. 43. Podobně tomu bylo v případě experimentu č. 2. Algoritmus HPA* PS tedy v některých případech může z vybraných algoritmů vrátit nejkratší cestu. Jak již bylo řečeno, vzhledem k tomu, že hierarchické metody obecně vrací suboptimální cesty, které nejsou nejkratší, je toto zajímavý výsledek, který může vést k výhodě použití hierarchických metod. Algoritmus HPA* PS má také nejnižší počet prohledaných uzlů, a tudíž provádí rychlejší hledání cesty za předpokladu, že má již vytvořené hierarchické prostředí. Konkrétně u mapy č. 3, která, jak již bylo řečeno, je méně výhodná pro A* algoritmus, je redukce uzlů nutných k prohledání výrazně nižší (téměř o 90%).



Obr. 43: Algoritmus (a) A* (b) A* PS (c) HPA* PS

Tab. 9: Výsledky experimentu č. 7

	Délka cesty	Vyhledávání a zpřesnění		Vyhledávání	Připojení startu a cíle
	[-]	[ms]	[ms]	Expandované uzly	
Mapa č. 1					
A*	54.11	4.00	0.00	547	0
A* PS	52.77	3.00	0.00	547	0
HPA* PS	54.86	0.00	0.00	26	30
Mapa č. 2					
A*	217.89	11.00	0.00	3770	0
A* PS	208.57	10.00	0.00	3770	0
HPA* PS	207.77	0.00	0.00	26	2831
Mapa č. 3					
A*	172.53	24.00	0.00	5467	0
A* PS	167.81	28.00	0.00	5467	0
HPA* PS	170.60	0.00	0.00	49	564

6 ZÁVĚR

Cílem diplomové práce bylo provést analýzu přístupů k plánování cesty s následným zaměřením na hierarchické přístupy. Vybrané hierarchické metody měly být dále implementovány a podrobeny srovnávacím a vyhodnocovacím experimentům.

První část diplomové práce se věnovala obecnému popisu problému plánování cesty, možným způsobům reprezentace výchozího prostředí mapy a základním algoritmům hledání cesty. V další části byly představeny hierarchické přístupy k plánování cesty a popsány jednotlivé hierarchické metody. Praktická část spočívala v implementaci vybraných algoritmů hledání cesty a vytvoření simulačního prostředí pro provedení srovnávacích a vyhodnocovacích experimentů. Pro implementaci bylo vybráno 5 algoritmů, a to algoritmy A*, A* PS, HPA*, HPA* PS a HPA* LEC. Algoritmy A* a A* PS představují zástupce klasických metod plánování cesty. Algoritmy HPA*, HPA* PS a HPA* LEC reprezentují hierarchické přístupy pro hledání cesty. Navržené simulační prostředí umožňuje načítání vstupních souborů formátu *.map* z benchmarku MovingAI, který obsahuje sadu testovacích prostředí, na nichž byly jednotlivé algoritmy spouštěny.

K porovnání vlastností jednotlivých algoritmů bylo provedeno celkem 7 experimentů. Jednotlivé experimenty zahrnovaly porovnání efektivity hierarchického a klasického přístupu plánování cesty, vliv rozdílného počátečního zpracování hierarchie na běh algoritmu, srovnání charakteru výsledných cest pro jednotlivé algoritmy a vliv velikosti vstupní mapy na efektivitu jednotlivých přístupů. V rámci experimentů byla zaznamenávána délka nalezené cesty pro jednotlivé algoritmy, počet expandovaných uzlů, tedy výpočetní náročnost, a doba běhu jednotlivých algoritmů. Ta byla zkoumána jak z hlediska doby pro samotné vyhledávání a zpřesňování cesty, tak z hlediska tvorby hierarchie, která je u hierarchických metod stěžejní.

Hierarchické metody jsou charakteristické tím, že je v počáteční fázi vytvářeno hierarchické prostředí výchozí mapy, které je poměrně časově náročné. Z toho důvodu vykazovaly pro počáteční běh algoritmu klasické metody lepší výsledky. Zmíněné prostředí hierarchie je však nutné vytvořit pouze jednou, a to na počátku vyhledávání. Následné hierarchické vyhledávání na nejvyšším abstraktním levelu dosahovalo již výrazně lepších výsledků než klasické přístupy, a to jak z hlediska časového, tak z hlediska paměťové a výkonnostní náročnosti, kdy v rámci abstrakce dochází ke značné redukci počtu prohledaných uzlů. Charakteristickou vlastností hierarchických metod je, že na rozdíl od klasické metody A* vrací suboptimální cesty, tedy cesty, které nejsou nejkratší. Rozdíl mezi optimální a suboptimální délkou trasy byl z podle dosažených výsledků testu minimální. Srovnáním charakterů cest pro jednotlivé algoritmy došlo k zajímavému zjištění, že rozšiřující metoda HPA* PS dokáže v některých případech vrátit nejkratší cestu ze všech testovaných algo-

ritmů. To může vést k další výhodě použití hierarchických metod. Při zkoumání vlivu rozdílného počátečního hierarchického zpracování bylo zjištěno, že pokud rozdělíme původní prostředí mapy na menší podoblasti a do více abstraktních levelů, bude doba tvorby počáteční hierarchie sice o něco časově náročnější, ale naopak každé další hledání bude již v takto vytvořeném abstraktním prostředí značně kratší. Stejně tak budou výrazně menší nároky na výkon z důvodu významné redukce prohledaných uzlů. To je způsobeno počátečním připojováním startovacího a cílového bodu do jednotlivých hierarchických levelů, kdy s většími počátečními podoblastmi je algoritmus nucen prohledat více uzlů základního rozlišení mapy a méně úrovní hierarchie. V rámci porovnání vlivu velikosti vstupního prostředí na vyhledávání pomocí klasického a hierarchického přístupu bylo zjištěno, že s rostoucí velikostí mapy roste velmi výrazně doba vyhledávání a počet expandovaných uzlů pro algoritmus A*. To představuje obrovské paměťové a výkonnostní nároky, které dokázaly hierarchické metody velmi významně zredukovat. Poslední srovnání uvádí výhodu použití algoritmu HPA* LEC, který snižuje náročnost tvorby hierarchického prostředí tím, že nevytváří kompletní hierarchické prostředí na počátku, ale sestavuje si ho až v samém průběhu vyhledávání podle toho, kterým směrem prohledávání postupuje. Tento přístup tak vedl k redukci celkového běhu algoritmu ve srovnání s algoritmem HPA*, a to ve všech testovaných případech.

7 SEZNAM POUŽITÉ LITERATURY

[1] Algfoor, Z. A., Sunar, M. S. a Kolivand, H. A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. *Int. J. Comput. Games Technol.* London, GBR: Hindawi Limited. jan 2015, sv. 2015. DOI: 10.1155/2015/736138. ISSN 1687-7047. Dostupné z: https://doi.org/10.1155/2015/736138.

[2] Amir, O., Sharon, G. a Stern, R. Multi-agent pathfinding as a combinatorial auction. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[3] Bartozel, Z. *Plánování cesty v reálném čase*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav automatizace a informatiky. Dostupné z: http://hdl.handle.net/11012/179055.

[4] Sturtevant, N. R. Benchmarks for Grid-Based Pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*. červen 2012, sv. 4, č. 2, s. 144–148. DOI: 10.1109/TCIAIG.2012.2197681. ISSN 1943-0698.

[5] BillWagner. *A tour of C# - Overview*. [online]. [cit. 2022-05-12]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[6] Botea, A., Bouzy, B., Buro, M., Bauckhage, C. a Nau, D. Pathfinding in Games. In:. Leden 2013, s. 21–31. DOI: 10.4230/DFU.Vol6.12191.21. ISBN 978-3-939897-62-0.

[7] Botea, A., Müller, M. a Schaeffer, J. Near optimal hierarchical pathfinding. *J. Game Dev.* Citeseer. 2004, sv. 1, č. 1, s. 1–30.

[8] Brondani, J. R., Lima Silva, L. A. de, Zacarias, E. a de Freitas, E. P. Pathfinding in hierarchical representation of large realistic virtual terrains for simulation systems. *Expert Systems with Applications*. 2019, sv. 138, s. 112812. DOI: https://doi.org/10.1016/j.eswa.2019.07.029. ISSN 0957-4174. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0957417419305081.

[9] Buniyamin, N., Sariff, N., Wan Ngah, W. a Mohamad, Z. Robot global path planning overview and a variation of ant colony system algorithm. *International journal of mathematics and computers in simulation*. Citeseer. 2011, sv. 5, č. 1, s. 9–16.

[10] Čápka, D. Lekce 1 - Úvod do C# a .NET frameworku. [online]. [cit. 2022-05-12]. Dostupné z: https://www.itnetwork.cz/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework.

[11] Čížek, L. *Navigace robotu pomocí grafových algoritmů*. Brno, 2011. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav automatizace a informatiky. [online] [cit. 2022-05-10]. Dostupné z: http://hdl.handle.net/11012/17172.

[12] Daniel, K., Nash, A., Koenig, S. a Felner, A. Theta*: Any-Angle Path Planning on Grids. *J. Artif. Intell. Res. (JAIR)*. Leden 2014, sv. 39. DOI: 10.1613/jair.2994.

[13] Demyen, D. a Buro, M. Efficient triangulation-based pathfinding. In: *Aaai*. 2006, sv. 6, s. 942–947.

[14] Dvořák, J. a Březina, T. *Algoritmy umělé inteligence*. 2013. Studijní opora. Vysoké učení technické v Brně, Fakulta strojního inženýrství.

[15] Ferguson, D. a Stentz, A. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*. Únor 2006, sv. 23, s. 79 – 101. DOI: 10.1002/rob.20109.

[16] Foudil, C., Noureddine, D., Sanza, C. a Duthen, Y. Path finding and collision avoidance in crowd simulation. *Journal of Computing and Information Technology*. Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu. 2009, sv. 17, č. 3, s. 217–228. DOI: 10.2498/cit.1000873.

[17] Harabor, D. a Botea, A. Hierarchical path planning for multi-size agents in heterogeneous environments. In: *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*. Prosinec 2008, s. 258–265. DOI: 10.1109/CIG.2008.5035648. ISBN 9781424429745. 2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008 ; Conference date: 15-12-2008 Through 18-12-2008.

[18] Hart, P. E., Nilsson, N. J. a Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. IEEE. 1968, sv. 4, č. 2, s. 100–107. DOI: 10.1109/TSSC.1968.300136.

[19] High Speed Priority Queue. *GitHub*. [online]. [cit. 2022-05-13]. Dostupné z: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp.

[20] Holte, R. C., Perez, M. B., Zimmer, R. M. a MacDonald, A. J. Hierarchical A*: Searching abstraction hierarchies efficiently. In: Citeseer. *AAAI/IAAI, Vol. 1*. 1996, s. 530–535.

[21] Isaac Computer Science. *Isaac Computer Science*. [online]. [cit. 2022-05-09]. Dostupné z: https://isaaccomputerscience.org/concepts/dsa_search_a_star?amp;stage=all&examBoard=all&stage=all.

[22] Jansen, M. a Buro, M. HPA* enhancements. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2007, sv. 3, č. 1, s. 84–87.

[23] Jones, M. *Artificial Intelligence: A Systems Approach: A Systems Approach*. Jones & Bartlett Learning, 2008. ISBN 9780763773373. Dostupné z: https://books.google.cz/books?id=-YJingEACAAJ.

[24] Karypis, G. a Kumar, V. Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*. 1998, sv. 48, č. 1, s. 96–129. DOI: https://doi.org/10.1006/jpdc.1997.1404. ISSN 0743-7315. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0743731597914040.

[25] Klobušníková, Z. *Plánování cesty mobilního robotu*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav automatizace a informatiky. [online]. [cit. 2022-05-10]. Dostupné z: http://hdl.handle.net/11012/81778.

[26] Koefoed Hansen, A. a Brodal, G. S. *Representations for path finding in planar environments*. 2012. Disertační práce. Citeseer.

[27] Maňáková, L. *Pokročilé metody plánování cesty mobilního robotu*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav automatizace a informatiky. [online]. [cit. 2022-05-10]. Dostupné z: http://hdl.handle.net/11012/191857.

[28] Nash, A., Daniel, K., Koenig, S. a Felner, A. Theta^*: Any-angle path planning on grids. In: *AAAI*. 2007, sv. 7, s. 1177–1183.

[29] Papadopoulou, E. a Lee, D. The L1 Voronoi Diagram Of Segments And Vlsi Applications. *International Journal of Computational Geometry Applications*. Prosinec 2002, sv. 11. DOI: 10.1142/S0218195901000626.

[30] Pathfinding Benchmarks. [online]. [cit. 2022-05-13]. Dostupné z: https://movingai.com/benchmarks/.

[31] Pelechano, N. a Fuentes, C. Hierarchical path-finding for Navigation Meshes (HNA∗). *Computers Graphics*. 2016, sv. 59, s. 68–78. DOI: https://doi.org/10.1016/j.cag.2016.05.023. ISSN 0097-8493. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0097849316300668.

[32] Rahmani, V. a Pelechano, N. Improvements to hierarchical pathfinding for navigation meshes. In:. Listopad 2017, s. 1–6. DOI: 10.1145/3136457.3136465.

[33] ŠEDA, M. Teorie graf. *Vysoké učení technické v Brně, Fakulta Strojního inženýrství, Ústav automatizace a informatiky, Brno*. 2003.

[34] Sekáč, O. *Plánování cesty pro více robotů*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav automatizace a informatiky. [online]. [cit. 2022-05-10]. Dostupné z: http://hdl.handle.net/11012/191861.

[35] Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N. et al. Path planning: A 2013 survey. In: IEEE. *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*. 2013, s. 1–8.

[36] Sturtevant, N. a Buro, M. Partial pathfinding using map abstraction and refinement. In: *AAAI*. 2005, sv. 5, s. 1392–1397.

[37] Stylet. *GitHub*. [online]. [cit. 2022-05-12]. Dostupné z: https://github.com/canton7/Stylet.

[38] Vermette, J. A survey of path-finding algorithms employing automatic hierarchical abstraction. *Journal of the Association for Computing Machinery*. 2011, sv. 377, s. 383.

[39] Wang, H., Qi, X., Lou, S., Jing, J., He, H. et al. An Efficient and Robust Improved A* Algorithm for Path Planning. *Symmetry*. 2021, sv. 13, č. 11. DOI: 10.3390/sym13112213. ISSN 2073-8994. Dostupné z: https://www.mdpi.com/2073-8994/13/11/2213.

[40] adegeo. *What is Windows Presentation Foundation - WPF .NET*. [online]. [cit. 2022-05-12]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/.

[41] Zhang, H.-y., Lin, W.-m. a Chen, A.-x. Path planning for the mobile robot: A review. *Symmetry*. Multidisciplinary Digital Publishing Institute. 2018, sv. 10, č. 10, s. 450.