

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

PLÁNOVANÍ CESTY MOBILNÍHO ROBOTU

MOBILE ROBOT PATH PLANNING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Zuzana Klobušníková
AUTHOR

VEDOUCÍ PRÁCE — RNDr. Jiří Dvořák, CSc.
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Studentka:	**Bc. Zuzana Klobušníková**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**RNDr. Jiří Dvořák, CSc.**
Akademický rok:	2017/18

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:

Plánování cesty mobilního robotu

Stručná charakteristika problematiky úkolu:

Úkolem systému pro plánování cesty robotu je najít cestu z počáteční do cílové pozice tak, aby se robot nedostal do kolize se známými překážkami a aby byla optimalizována nějaká kriteriální funkce.

Cíle diplomové práce:

Analyzovat přístupy k plánování cesty mobilního robotu.
Implementovat vybrané metody plánování cesty.
Provést a vyhodnotit ověřovací a srovnávací experimenty.

Seznam doporučené literatury:

MAC, T. T. et al. Heuristic Approaches in Robot Path Planning: A Survey. Robotics and Autonomous Systems, vol. 86, 2016, pp. 13-28.

DUCHOŇ, F. et al. Path Planning with Modified A Star Algorithm for a Mobile Robot. Procedia Engineering, vol. 96, 2014, pp. 59-69.

DAKULOVIĆ, M. et al. Efficient Interpolated Path Planning of Mobile Robots Based on Occupancy Grid Maps. IFAC Proceedings Volumes, vol. 45, issue 22, 2012, pp. 349-354.

HARABOR, D.; GRASTIEN, A. Online Graph Pruning for Pathfinding on Grid Maps. In AAAI'11 Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011, pp. 1114-1119.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2017/18

V Brně, dne

L. S.

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

ABSTRAKT

Táto diplomová práca sa zaoberá plánovaním cesty robota pomocou vybraných grafových algoritmov umelej inteligencie. Teoretická časť popisuje základné metódy plánovania cesty robota. Ďalej sa bližšie venuje grafovým algoritmom. Praktická časť práce sa zaoberá implementáciou vybraných grafových algoritmov, vytvorením simulačného prostredia v jazyku Python, popisom a vyhodnotením experimentov.

ABSTRACT

This master thesis deals with the planning of the robot's path using selected graph algorithms of artificial intelligence. The theoretical part describes the basic methods of planning a robot's path. It is related to the graph algorithm more closely. The practical part deals with implementation of selected graph algorithms, creation of simulation environment in Python, description and evaluation of experiments.

KLÍČOVÁ SLOVA

Plánovanie cesty, mobilný robot, grafové algoritmy

KEYWORDS

Path planning, mobile robot, graph algorithms

BIBLIOGRAFICKÁ CITACE

KLOBUŠNÍKOVÁ, Z. *Plánování cesty mobilního robotu,* Brno, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018, 62 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc.

PODĚKOVÁNÍ

Ďakujem vedúcemu diplomovej práce RNDr. Jiřímu Dvořákovi, CSc. za cenné rady a pripomienky.

ČESTNÉ PROHLÁŠENÍ

Prehlasujem, že táto práca je mojím pôvodným dielom, spracovala som ju samostatne pod vedením RNDr. Jiřího Dvořáka, CSc. a s použitím literatúry uvedenej v zozname literatúry.

V Brně dne 24. 5. 2018 ……

Bc. Zuzana Klobušníková

OBSAH

1 ÚVOD 15

2 PLÁNOVANIE CESTY 17
2.1 Metódy navigácie robota 17
2.2 Metódy spracovania pracovného priestoru 18
2.2.1 Metóda rozkladu do buniek 18
2.2.2 Metóda mapy ciest 19
2.2.3 Pravdepodobnostné mapy ciest 20
2.2.4 Pravdepodobnostné stromy 21
2.2.5 Potenciálové polia 22
2.3 Metódy plánovania cesty 22
2.3.1 Rojová inteligencia 22
2.3.2 Genetické algoritmy 24

3 GRAFOVÉ ALGORITMY 25
3.1 Dijkstrov algoritmus 26
3.2 Algoritmus A* 28
3.3 Algoritmus NBA* 33
3.4 Jump Point Search algoritmus (JPS) 35
3.4.1 Body skoku (Jump points) 35
3.4.2 Prerezávacie pravidlá 39
3.4.3 Popis algoritmu 39

4 POPIS APLIKÁCIE 43
4.1 Členenie programu 44
4.2 Návrh užívateľského rozhrania 45

5 VÝSLEDKY EXPERIMENTOV 49

6 ZÁVER 57

7 ZOZNAM POUŽITEJ LITERATÚRY 59

1 ÚVOD

Táto práca sa zaoberá plánovaním cesty mobilného robota. Plánovanie cesty robota prebieha za určitých obmedzujúcich podmienok ako sú napr. smery pohybu robota alebo čas potrebný pre nájdenie riešenia. Výsledkom plánovania je nájdenie cesty v priestore, ktorá nie je v kolízii s prekážkami.

Ak by sme mali zhodnotiť posledné tri desaťročia, plánovanie cesty sa stalo rozhodujúcou a produktívne významnou oblasťou robotiky. V priebehu niekoľkých posledných rokov nastal v oblasti plánovania ciest obrovský rozvoj. V súčasnosti sa môžeme stretnúť s novými metódami, ktoré sú schopné nájsť riešenie zložitých problémov zahrňujúcich roboty s mnohými stupňami voľnosti. Výber správnej metódy môže byť ovplyvnený ako typom robota, tak aj optimalitou nájdenej cesty. S plánovaním ciest sa dnes môžeme stretnúť nielen v priemyslovej automatizácii, ale aj v lekárstve či domácnostiach.

Cieľom tejto práce je popísať metódy navigácie robota, metódy spracovania priestoru, základné metódy plánovania ciest a vybrané grafové algoritmy, ktoré patria do skupiny klasických metód umelej inteligencie. Ďalej implementovať zvolené algoritmy, navrhnúť simulačné prostredie a previesť porovnávacie experimenty. Pre implementáciu je zvolený Dijkstrov algoritmus, algoritmus A*, NBA* (*New Bidirectional A**), JPS (*Jump Point Search*) a je vysvetlený princíp týchto algoritmov. Ďalšia časť práce popisuje navrhnuté simulačné prostredie a v experimentálnej časti sú porovnané výsledky získané týmito algoritmami, a to z hľadiska časovej náročnosti, použitej heuristickej funkcie a počtu expandovaných uzlov pri ceste zo štartového do cieľového uzlu.

Algoritmy použité v diplomovej práci pracujú v statickom dvojrozmernom priestore, v ktorom sú prekážky vopred známe. Implementované algoritmy musia byť schopné nájsť optimálne riešenie v čo najkratšom čase.

2 PLÁNOVANIE CESTY

Plánovanie cesty je termín známy hlavne z oblasti robotiky ako súčasť navigácie autonómneho robota. Výsledkom plánovania cesty je postup, ako presunúť robota z jedného miesta do druhého, pričom nesmie dôjsť ku kolízii so známymi prekážkami alebo iným robotom či zariadením nachádzajúcim sa v rovnakom prostredí.

Najčastejšie rozlišujeme dva prístupy pre plánovanie cesty robota, globálne a lokálne plánovanie. Globálne plánovanie cesty je založené na známom prostredí a je vhodné pre nájdenie optimálnej cesty. Opakom je lokálne plánovanie cesty, kde robot adekvátne reaguje na neznáme alebo dynamické prekážky. Môžeme povedať, že robot sa rozhoduje počas pohybu. V prípade kombinácie oboch metód je možné dosiahnuť lepších výsledkov. [1; 2; 3]

Pre plánovanie cesty robota existuje pomerne veľké množstvo metód, kde každú z týchto metód môžeme rozdeliť na dve časti. V prvej časti sa prevádza predspracovanie, kde sa pracovný priestor diskretizuje a následne sa prevedie na konfiguračný priestor. Konfiguračný priestor tvoria všetky konfigurácie robota v pracovnom priestore, kde robot plní dané úlohy. Konfigurácia robota je daná vektorom, ktorého zložky určujú polohu a orientáciu robota. V konfiguračnom priestore sú prekážky zväčšené a robot je reprezentovaný bodom. Po diskretizácii pracovného priestoru je konfiguračný priestor popísaný pomocou grafu, mriežky alebo funkcie. Druhou časťou plánovania cesty je opytovacia časť, kde je použitá niektorá metóda pre hľadanie cesty medzi počiatočnou a cieľovou pozíciou robota. Použitou metódou môže byť niektorá z klasických metód alebo metód umelej inteligencie.

2.1 Metódy navigácie robota

Navigácia mobilných robotov je definovaná ako určenie ich pozície v priestore a nájdenie vhodnej cesty z počiatočného bodu do bodu cieľového. Navigáciu môžeme rozdeliť do troch hlavných skupín [4; 5]:

- Lokalizácia – proces určenia aktuálnej pozície robota,
- Učenie mapy – proces zapamätávania dát získaných behom prevádzky robota,
- Plánovanie cesty – proces výberu akcie na dosiahnutie cieľa vzhľadom k súčasnej polohe tak, aby nedošlo ku kolízii s prekážkou.

V praxi sa často stretávame so vzájomným spojením lokalizácie a učenia mapy. V literatúre toto spojenie nájdeme pod názvom SLAM (simultánna lokalizácia a mapovanie). Tento problém si môžeme vysvetliť ako vytvorenie mapy neznámeho prostredia, v ktorom sa robot pohybuje a zároveň určovanie jeho polohy na základe tejto mapy. [3]

2.2 Metódy spracovania pracovného priestoru

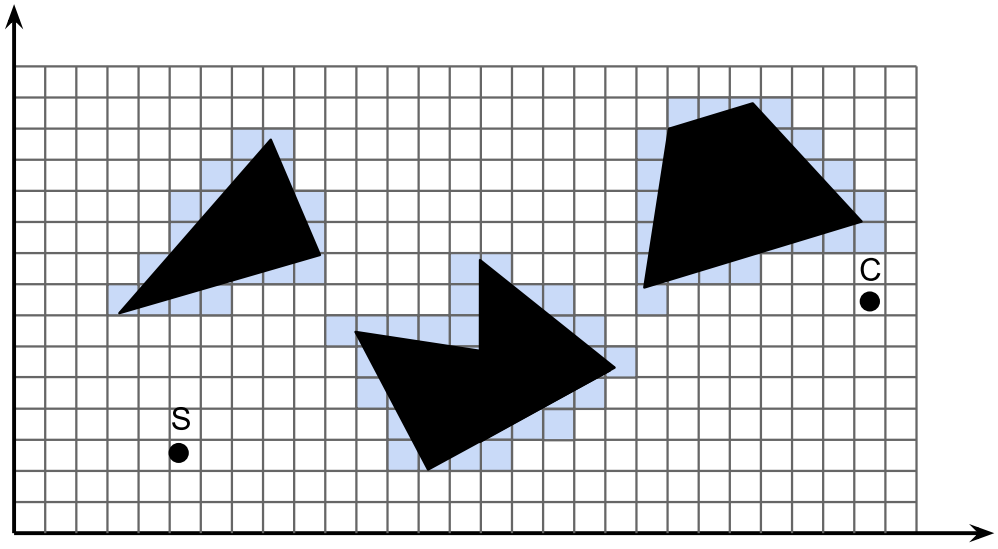
Pre spracovanie pracovného priestoru existuje pomerne veľké množstvo metód. K týmto metódam existuje niekoľko modifikácii, preto budú v tejto kapitole popísané len tie základné. Informácie uvedené v tejto kapitole sú čerpané hlavne zo zdrojov [5; 6; 7; 8; 9].

2.2.1 Metóda rozkladu do buniek

V tomto prípade je pracovný priestor rozložený do buniek určitého tvaru a veľkosti. U každej bunky sa určí, či obsahuje prekážku alebo voľný priestor. Na základe týchto informácii sa vytvorí grafová štruktúra, kde bunky neobsahujúce prekážku tvoria vrcholy a hrany tvoria spojnice medzi nimi. V závislosti na tom, ako chceme pracovný priestor rozdeliť, môžeme použiť aproximatívny alebo exaktný rozklad.

Aproximatívny rozklad

Táto metóda rozkladá celý pracovný priestor na bunky rovnakého tvaru. Tvar buniek je najčastejšie štvorcový, ale nemusí byť konštantný. Každá bunka je označená či je prázdna alebo obsadená prekážkou. Ak použijeme bunky rovnakej veľkosti, bunka bude označená za obsadenú ak obsahuje nejakú časť prekážky. Presnosť aproximácie sa odvíja od veľkosti buniek. Čím sú bunky menšie, tým je rozklad presnejší, ale na druhú stranu je zvýšená výpočtová a pamäťová náročnosť. Mriežka reprezentuje grafovú štruktúru, ktorej uzly odpovedajú bunkám neobsadeným prekážkami a sú spojené hranami s ich susedmi (tých môže byť 4 alebo 8, podľa obmedzení pohybu robota).

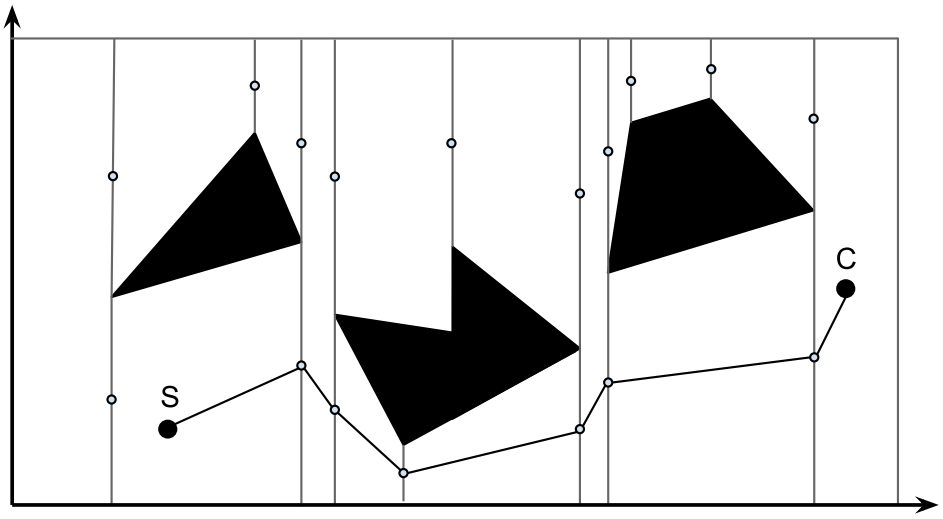


Obr. 1: Ukážka aproximatívneho rozkladu

Exaktný rozklad

Princíp tejto metódy spočíva v rozdelení priestoru do množiny neprekrývajúcich sa buniek. Tvar buniek je jednoduchý, často ide o trojuholník alebo lichobežník a to s ohľadom na výpočtovú náročnosť. Uzly grafovej štruktúry sú tvorené bodmi prechodu,

ktoré vznikajú na hranici medzi susednými bunkami. V takto vytvorenej grafovej štruktúre sa hľadá cesta medzi štartovým a cieľovým bodom, ako je znázornené na obr.2. Táto metóda je kompletná, pretože ak cesta existuje, vždy je nájdená.



Obr. 2: Ukážka exaktného rozkladu

2.2.2 Metóda mapy ciest

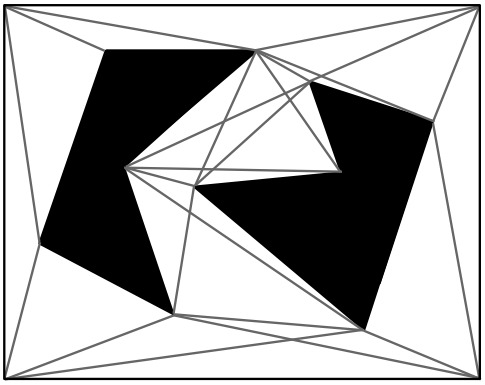
Metóda mapy ciest vytvára grafovú štruktúru, ktorá reprezentuje voľný pracovný priestor. Vrcholy tvoria body v pracovnom priestore (v závislosti na použitej metóde) a hrany predstavujú cesty, po ktorých sa môže robot pohybovať. Ak budú do grafu ako ďalšie vrcholy pridané počiatočná a cieľová pozícia robota, vytvorili sme cestovnú mapu. Ďalej je táto mapa použitá pre hľadanie vhodnej cesty pri použití niektorého známeho algoritmu.

Metódu mapy ciest označuje ako metódu úplnú, čo znamená, že ak cesta existuje, bude nájdená. Ďalej platí, že nájdená cesta je optimálna. Metódu delíme na deterministickú a pravdepodobnostnú. Medzi deterministické metódy patrí napr. graf viditeľnosti a Voronoiov diagram.

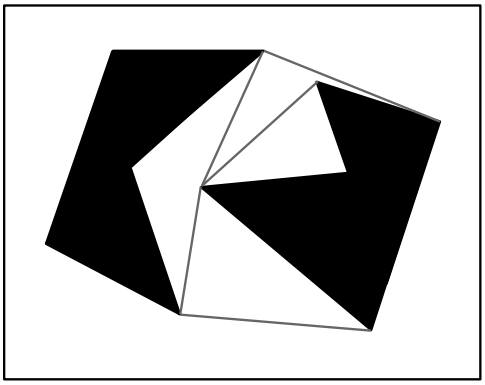
Graf viditeľnosti

Jednotlivé uzly grafu odpovedajú vrcholom prekážok, počiatočnej a cieľovej pozície robota. Dvojica uzlov je prepojená hranou len v tom prípade, že medzi nimi nie je prekážka. Graf viditeľnosti je použiteľný len v prípade, že pracovný priestor obsahuje prekážky v tvare polygónu. Nepolygónne prekážky musia byť nahradené polygónom, čo ale zvyšuje počet vrcholov a tým aj čas potrebný pre vytvorenie tohto grafu.

Časovú náročnosť vytvárania grafu je možné znížiť vytvorením redukovaného grafu viditeľnosti, tzv. grafu dotyčníc. V tomto prípade dochádza k odstráneniu nadbytočných hrán a to tých, ktoré nie sú dotyčnicami prekážok. Môžeme povedať, že odstránime hrany, ktorých predĺženie je vo voľnom priestore [10]. Pre tento graf prekážky v priestore nemusia byť len polygonálneho tvaru.



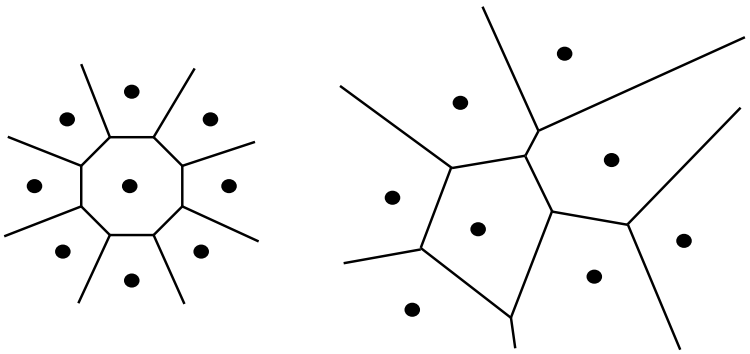
Obr. 3: Graf viditeľnosti (upravené podľa [11])



Obr. 4: Graf dotyčníc (upravené podľa [11])

Voronoiov diagram

Ide o rovinnú geometrickú štruktúru tvorenú bodmi, ktoré majú rovnakú vzdialenosť od dvoch alebo viacerých vrcholov prekážok. Body na rozhraní dvoch území tvoria hrany diagramu, po ktorých sa môže robot pohybovať v dostatočnej vzdialenosti od prekážky. Z toho diagramu je následne vytvorený graf, kde uzly odpovedajú bodom s rovnakou vzdialenosťou od troch alebo viacerých vrcholov prekážok a hrany grafu odpovedajú bodom v rovnakej vzdialenosti od dvoch vrcholov prekážky. Do grafu je potrebné zahrnúť tiež počiatočný a cieľový bod robota. [8]



Obr. 5: Voronoiov diagram (upravené podľa [12])

2.2.3 Pravdepodobnostné mapy ciest

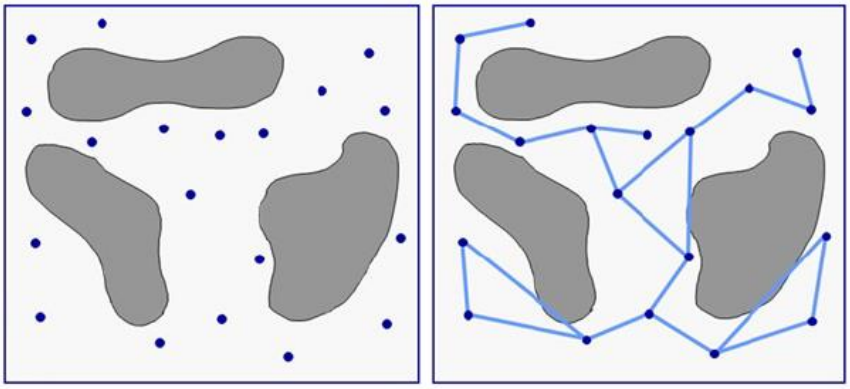
Samotný algoritmus tvoria dve časti, učiaca a opytovacia. V prvej časti sa vygeneruje mapa ciest a v druhej sa hľadá cesta v tomto grafe.

Generovanie grafu prebieha náhodným spôsobom a to tak, že vrcholy predstavujú náhodne vybrané konfigurácie robota, ktoré ale musia byť vo voľnom pracovnom priestore. Hrany grafu tvoria priame nekolízne spojnice vrcholov. Táto fáza sa opakuje a končí v prípade dostatočne popísaného voľného priestoru alebo ak je splnené nejaké kritérium ukončenie, napr. vyčerpanie času, pamäti. [11]

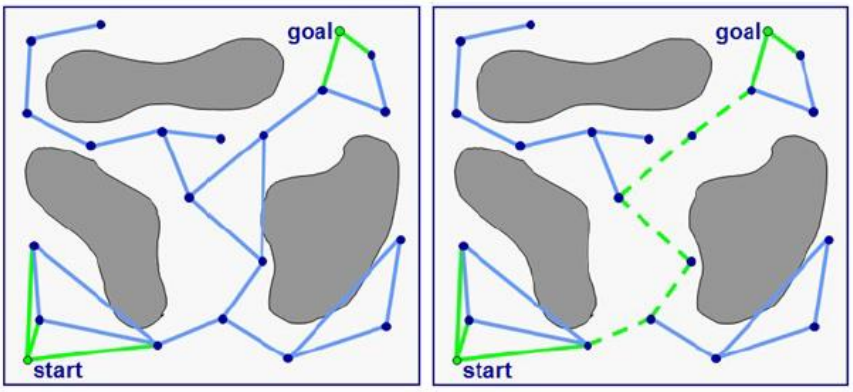
V druhej fáze je nutné do grafu napojiť počiatočný a cieľový vrchol, ktoré sa prepoja možnými cestami. Následne dôjde k hľadaniu cesty vo vytvorenom grafe medzi týmito vrcholmi. Ak cesta nebude nájdená, pracovný priestor nebol dostatočné pokrytý a je možné prvú fázu zopakovať. Ako uvádza práca [7], tento algoritmus je vhodný

predovšetkým pre prostredie so statickými prekážkami, kde je možné niekoľko krát opakovať fázu druhú, bez nutnosti opakovania prvej fázy.

Nevýhodou tejto metódy je, že v prípade prekážok, ktoré vytvárajú v priestore len úzky prechod, dochádza k rozpadu mapy na niekoľko neprepojených segmentov. Preto sa môže stať, že cesta medzi počiatočným a cieľovým vrcholom nebude nájdená z dôvodu, že jeden z vrcholov bude náležať inému segmentu.



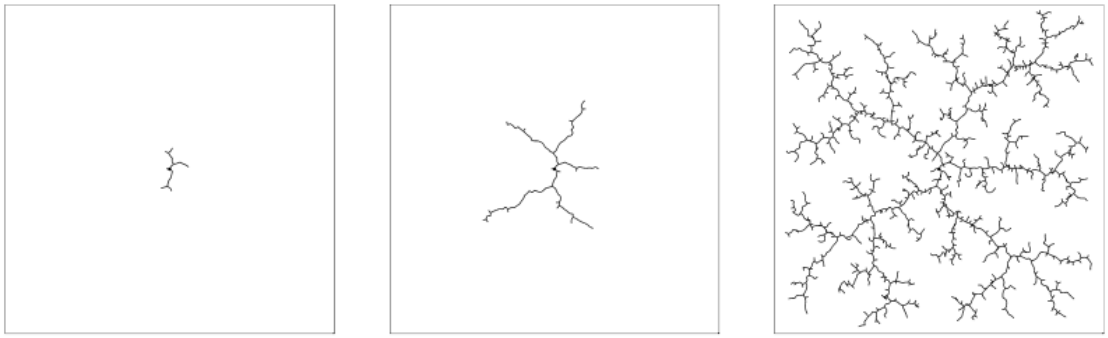
Obr. 6: Učiaca fáza [7]



Obr. 7: Opytovacia fáza [7]

2.2.4 Pravdepodobnostné stromy

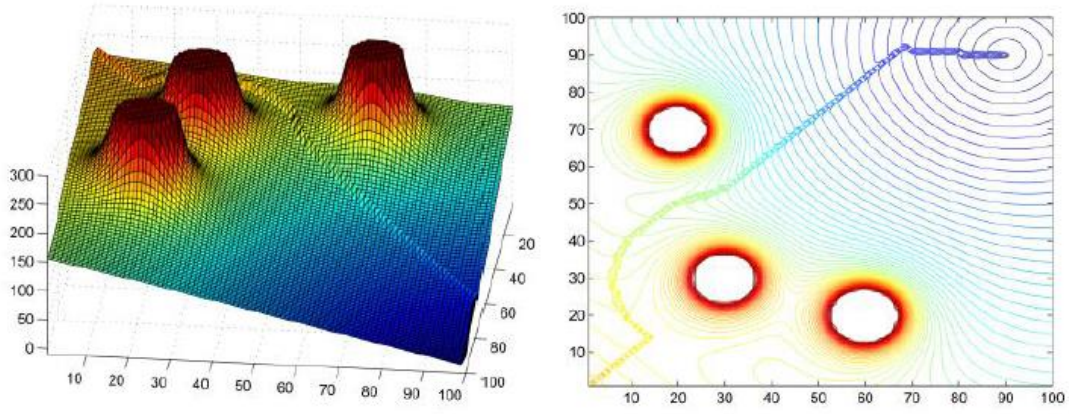
Metóda pravdepodobnostných stromov je alternatíva k pravdepodobnostným mapám ciest. Práve metóda pravdepodobnostných stromov zachováva prepojenie všetkých vrcholov v súvislom strome. Hlavnou myšlienkou je rýchle inkrementálne prehľadávanie konfiguračného priestoru, čím sa vytvára prehľadávací strom. V každej iterácii sa strom rozrastie o nový vrchol smerom k náhodne vygenerovanej konfigurácii robota. Hrany predstavujú jednotlivé akcie robota. Podrobnému popisu tejto metódy sa venuje jej autor v článku [13].



Obr. 8: Príklad prehľadávacieho stromu [14]

2.2.5 Potenciálové polia

Princíp tejto metódy spočíva v pokrytí pracovného priestoru potenciálovým poľom definovaným potenciálovou funkciou $\varphi(x,y)$. Štartová pozícia robota má vyšší potenciál než cieľová a prekážky majú potenciál vyšší než ich okolie. Sú obklopené odpudivým poľom, ktorého intenzita klesá so vzdialenosťou od prekážok. Robot sa v prostredí pohybuje v smere opačného gradientu potenciálovej funkcie. Možnosť uviaznutia robota v lokálnom minime predstavuje nevýhodu tejto metódy, ktorú ide riešiť inými prístupmi.



Obr. 9: Metóda potenciálových polí [15]

2.3 Metódy plánovania cesty

Po ukončení prvej fáze, čiže predspracovaní pracovného priestoru sa pre nájdenie cesty robota medzi štartovacou a cieľovou pozíciou aplikujú konkrétne plánovacie algoritmy. Výber vhodného algoritmu je závislý na konkrétnej úlohe a nárokoch na jej riešenie. O výbere hovoríme z toho dôvodu, že plánovacích algoritmov existuje veľké množstvo. Ďalej budú uvedené niektoré metódy umelej inteligencie, klasické metódy boli v tejto diplomovej práci zvolené k implementácii a podrobnejšie sa im venuje kapitola 3.

2.3.1 Rojová inteligencia

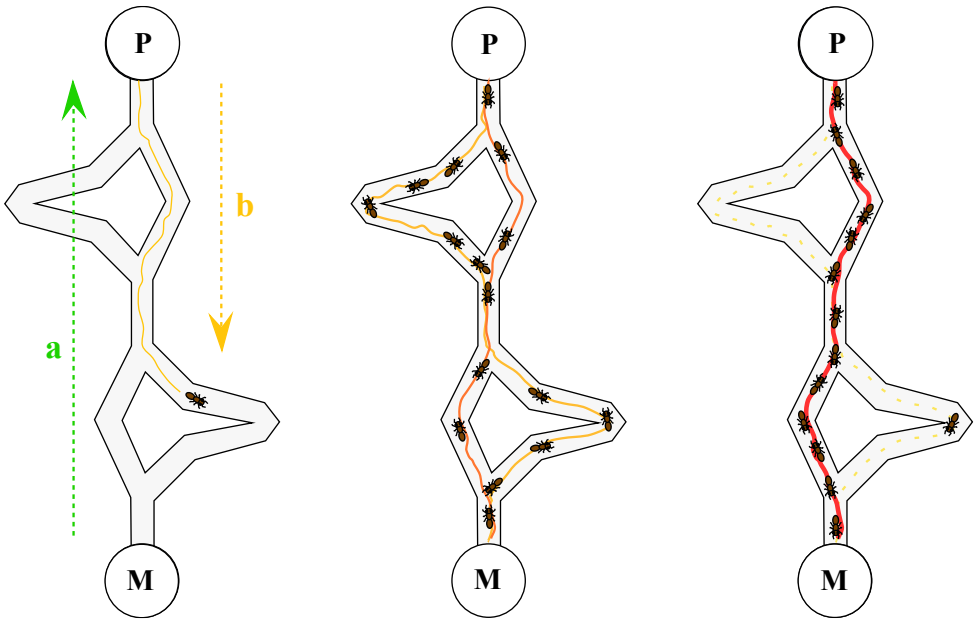
Rojová inteligencia vychádza z chovania živočíchov žijúcich v roji ako sú mravce, vtáky či ryby. Algoritmy rojovej inteligencie sú založené na vzájomnom ovplyvňovaní

jednoduchých agentov, ktorý tvoria roj. Komunikácia medzi agentami závisí od prirodzeného chovania živočíchov v prírode. Výhodou týchto algoritmov je schopnosť prehľadávať neznámy priestor a prispôsobiť sa zmenám v tomto priestore. Medzi najznámejšie algoritmy rojovej inteligencie patria mravčie algoritmy, včelie algoritmy a metóda rojenia častíc. Ďalej k algoritmom rojovej inteligencie patrí napríklad samoorganizujúci sa migračný algoritmus [16] a algoritmus svätojánskych mušiek [17].

Mravčie algoritmy (ACO - Ant Colony Optimization)

Patria do skupiny algoritmov rojovej inteligencie a ako napovedá názov, algoritmus je inšpirovaný chovaním mravcov, či už pri hľadaní potravy alebo stavbe mraveniska . Hlavnou myšlienkou mravčích algoritmov sú princípy samoorganizácie, ktoré u reálnych mravcov umožňujú vysoko koordinované chovanie a môžu byť použité pre populáciu umelých jedincov spolupracujúcich na riešení výpočtového problému. [8; 18]

Spolupráca je založená na nepriamej komunikácii v rámci kolónie umelých mravcov pomocou feromónových stôp. Rovnako ako v prírode majú umelé mravce pravdepodobnostné chovanie a tieto stopy im slúžia ako informácie pre riešenie problému. Behom vykonávania algoritmu sa feromónové stopy upravujú posilňovaním (pri prechode mravca) a odparovaním. Čím je vyššia hodnota feromónu, tým je vyššia pravdepodobnosť, že sa mravce budú držať tejto cesty. Z toho vyplýva, že hodnota feromónu na tejto ceste bude vzrastať, naopak ostatné cesty budú strácať na intenzite feromónu, čo môžeme vidieť na obr.10, kde *M* symbolizuje mravenisko a *P* potravu. Podrobnejší popis môžeme nájsť napr. v článku [19].



Obr. 10: Experiment dvojitého mostu [20]

Algoritmus umelých včelích kolónií (ABC - Artificial Bee Colony)
Algoritmus umelých včelích kolónií [21] je inšpirovaný včelami medonosnými a ich spôsobom hľadania potravy. Hlavnou myšlienkou algoritmu je samoorganizácia a delenie práce, ktoré je pre včely typické. Komunikácia medzi nimi prebieha prostredníctvom tanca a feromónov. Princíp algoritmu spočíva v spolupráci troch druhov agentov, spravodajky, rekrutky a priezkumnice, kde každý druh má svoju úlohu. Na začiatku je zvolená počiatočná populácia riešenia (zdrojov potravy). Podobne ako u mravčích algoritmov je kvalita každého zdroja ohodnotená fitness funkciou. Celý algoritmus prebieha v troch fázach až do ukončenia. Algoritmus končí, ak je nájdené najlepšie riešenie alebo ak vyprší čas. [22]

Rojenie častíc (PSO – Particle Swarm Optimization)
Rojenie častíc alebo tiež metóda optimalizácie rojom [23] je inšpirovaná chovaním roja vtákov a rýb. Agenti, ktorý medzi sebou spolupracujú sa nazývajú častice a každá častica má svoj vektor polohy a vektor rýchlosti. Vektor rýchlosti, teda smer a rýchlosť pohybu si agenti upravujú počas jednotlivých iterácií algoritmu podľa agenta, ktorý má doposiaľ najlepšiu skúsenosť v priestore. Častice sa pohybujú priestorom prípustných riešení a sú priťahované do sľubných miest na základe svojich skúseností alebo skúseností iných častíc. [21]

2.3.2 Genetické algoritmy

Genetické algoritmy (GA) patria do skupiny evolučných metód a používajú termíny ako jedinci, selekcia, kríženie a mutácia, ktoré sú známe z biológie. Populáciou rozumieme určitý počet jedincov, pričom pri riešení úloh reprezentuje jedinec konkrétne riešenie. Jedinec je tvorený reťazcom konečnej dĺžky (najčastejšie binárny reťazec) v ktorom sú zakódované informácie o riešení problému (hovoríme o chromozóme). Inicializačná fáza spočíva vo vytvorení počiatočnej populácie, ktorá môže byť vygenerovaná úplne náhodne. Prechod do novej populácie, ktorú tiež nazývame generácia, je spojený s výpočtom tzv. fitness funkcie, ktorá udáva kvalitu riešenia každého jedinca v populácii. Na vytvorení novej generácie sú vybrané jedince s ohľadom na hodnotu fitness funkcie. V priebehu algoritmu sa na nich aplikujú genetické operátory, spravidla sa jedná o kríženie a mutáciu. Tento postup sa opakuje až kým nie je splnená ukončovacia podmienka. [24]

Genetické algoritmy sa využívajú pre plánovanie cesty robota v diskrétnom aj spojitom prostredí. Voľba genetických operátorov je rozhodujúcim krokom pre konvergenciu algoritmu a článok [25] popisuje nový prístup pre riešenie problémov plánovania ciest.

3 GRAFOVÉ ALGORITMY

Grafové algoritmy slúžia k prehľadávaniu stavového priestoru, ktorý je reprezentovaný grafovou štruktúrou. Stavový priestor predstavuje množinu stavov a množinu pravidiel, ktorými je možné prechádzať medzi jednotlivými stavmi. Vrcholy grafu predstavujú stavy a hrany pravidlá. Ich plnením sa dostaneme z jedného stavu do druhého a výsledkom je nájdenie cesty v grafe.

Pre efektívne prehľadávanie stavového priestoru je výhodné využiť znalosti o riešenom probléme. Ak by bol stavový priestor príliš veľký a my by sme prehľadávali nesystematicky, zbytočne by sme prehľadávali stavy, ktoré by nás neviedli k cieli. Vzhľadom k využitiu týchto znalostí delíme metódy prehľadávania grafu na neinformované a informované. [26]

Neinformované metódy

Neinformované metódy nemajú žiadne znalosti o stavovom priestore, ktoré by pomohli rýchlejšiemu prehľadávaniu. Ako bolo spomenuté vyššie, prechádza sa veľké množstvo stavov kým nie je nájdené cieľové riešenie. Z toho vyplýva značná časová zložitosť, preto neinformované metódy nie sú vhodné pre zložitejšie úlohy.

Medzi tieto metódy patrí napr. slepé prehľadávanie do šírky (BFS) a do hĺbky (DFS). Oba tieto algoritmy pracujú so zoznamom neexpandovaných uzlov (OPEN) a zoznamom expandovaných uzlov (CLOSED). U algoritmu BFS sa expanduje najprv uzol s najmenšou hĺbkou a nájdené riešenie je najkratšou cestou vzhľadom k počtu hrán. Algoritmus DFS naopak expanduje ako prvé uzly s najväčšou hĺbkou, čo je spojené s nižšími časovými nárokmi na pamäť. Za nevýhodu tohto algoritmu považujeme neoptimálne riešenie, čo znamená, že nemusí byť nájdená najkratšia cesta. [27]

Informované metódy

Informované metódy využívajú určité znalosti o stavovom priestore obmedzujúce nadbytočné prehľadávanie stavového priestoru. Môžu byť založené na heuristickej funkcii, ktorá umožňuje odhadnúť ako ďaleko sa nachádzame od cieľa alebo na ohodnotení hrán. Uzol k expanzii je možné vybrať použitím ohodnotenia hrany, hodnoty heuristickej funkcie alebo oboch hodnôt. Čím je heuristická funkcia kvalitnejšia, tým bude prehľadávanie efektívnejšie.

Medzi najznámejšie informované metódy patrí Dijkstrov algoritmus a algoritmus A*. Práve z algoritmu A* vychádzajú mnohé ďalšie, ako napr. IDA*, D*, D* Lite, ARA*. V prípade algoritmov D* [28] hovoríme o algoritme typu replanning, ktoré sú vhodné pre plánovanie cesty robota v čiastočne známom, prípadne dynamickom prostredí. Algoritmus ARA* [29] patrí k algoritmom typu anytime, ktoré sú schopné nájsť riešenie v obmedzenom čase.

Do kategórie informovaných metód radíme tiež gradientný algoritmus (Hill-climbing) a algoritmus usporiadaného prehľadávania (best-first search).

3.1 Dijkstrov algoritmus

Dijkstrov algoritmus vymyslel holandský informatik Edsger W. Dijkstra, po ktorom má tento algoritmus názov, v roku 1959 [30]. Tento algoritmus nájde všetky najkratšie cesty zo zadaného uzlu do ostatných uzlov grafu. Predpokladom pre správne fungovanie Dijkstrovho algoritmu je nezáporné ohodnotenie hrán grafu. V prípade záporne ohodnotených hrán sa používa pomalší Bellman-Fordov algoritmus (Richard Bellman, Lester Ford, 1958).

Ohodnotenie hrán grafu predstavuje dĺžky hrán medzi jednotlivými uzlami, ktoré môžu reprezentovať napr. čas alebo náklady spojené s prechodom po hrane alebo dĺžku hrany v použitých jednotkách. Dĺžku najkratšej cesty zo zdroja *s* k vrcholu *v* označíme ako $d(v)$. Je zrejmé, že zdroj má dĺžku $d(s)=0$. Na začiatku algoritmu je pre všetky ostatné uzly hodnota $d(v)=\infty$, pretože cestu k týmto uzlom zatiaľ nepoznáme. Dijkstrov algoritmus si uchováva všetky uzly v prioritnej fronte, ktoré sú zoradené podľa vzdialenosti od zdroja. Algoritmus je založený na postupnom spresňovaní odhadu dĺžky najkratšej cesty medzi zdrojom a ostatnými uzlami. Spresňovanie prebieha v cykle, kým nie je prioritná fronta prázdna. V každom kroku sa vyberie uzol *u* s najvyššou prioritou, teda s najmenšou dĺžkou cesty a hovoríme, že algoritmus používa hladnú (greedy) stratégiu. Potom sa uzol zaradí do množiny *D*, ktorú tvoria spracované uzly. Pre susedov vybraného uzlu, ktorý sa nenachádzajú v množine *D*, sa overuje podmienka:

$$d[u] + l[u, v] < d[v], \tag{1}$$

kde $l(u,v)$ je dĺžka hrany medzi vybraným uzlom *u* a susedným uzlom *v*. Ak je podmienka splnená, danému susedovi je upravená hodnota dĺžky cesty $d(v)$ tak, že vo vzťahu (1) platí rovnosť. Pre rekonštruovanie cesty zo zdroja do uzlu *y* sa mu ako predchodca $p(v)$ priradí práve spracovávaný uzol *u*. Na začiatku sú samozrejme hodnoty predchodcov nastavené na *None*. Ak sú v cykle preskúmaný všetci susedia aktuálneho uzlu, algoritmus pokračuje výberom nového uzlu z prioritnej fronty a celý postup sa opakuje.

Algoritmus 1 Dijkstrov algoritmus

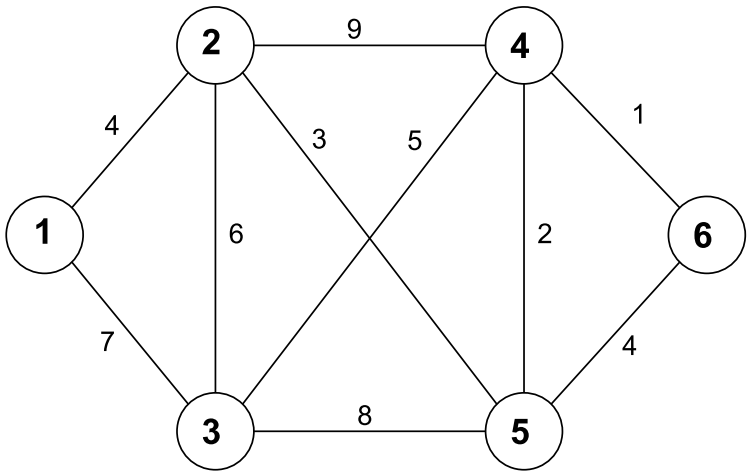
```
DIJKSTRA(G, s)
 1  // Inicializácia
 2  for each vertex v ∈ V_G
 3       distance[v] ← ∞
 4       parent[v] ← None
 5       Q ← v
 6
 7  distance[s] ← 0
 8
 9  while Q ≠ ∅
10       u ← EXTRACT-MIN(Q)
11       for each neighbor v of u
12            if distance[v] > distance[u] + length[u, v]
13                 distance[v] ← distance[u] + length[u, v]
14                 parent[v] ← u
15
16  return distance[], parent[]
```

Algoritmus 1

Dijkstrov algoritmus končí vyprázdnením prioritnej fronty. Tiež je možné algoritmus ukončiť skôr a to pridaním podmienky, ktorá kontroluje či práve expandovaný uzol nie je uzlom cieľovým.

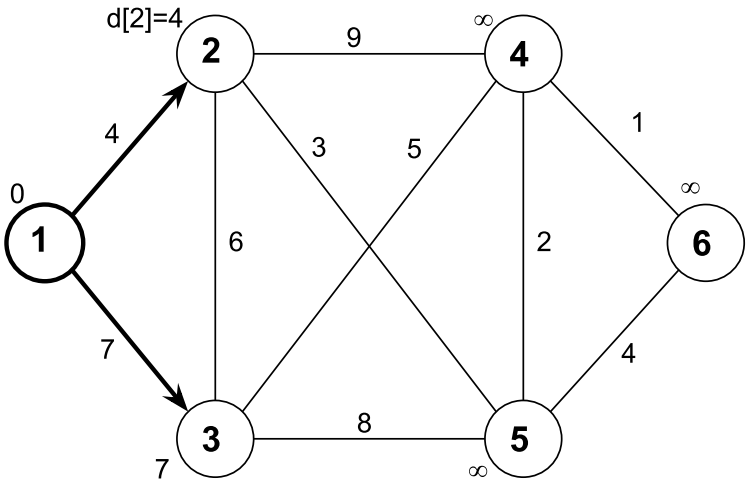
Od implementácie prioritnej fronty závisí časová zložitosť algoritmu. Pritoritná fronta môže byť implementovaná tiež pomocou binárnej alebo Fibonacciho haldy. V úplne najjednoduchšom prípade je zložitosť algoritmu $O(|V^2|)$, kde V je počet vrcholov grafu. [12]

Na nasledujúcom jednoduchom príklade si ukážeme princíp Dijkstrovho algoritmu. V grafe na obr.11 chceme nájsť najkratšiu cestu z vrcholu 1 do vrcholu 6.



Obr. 11: Graf pre hľadanie cesty Dijkstrovym algoritmom

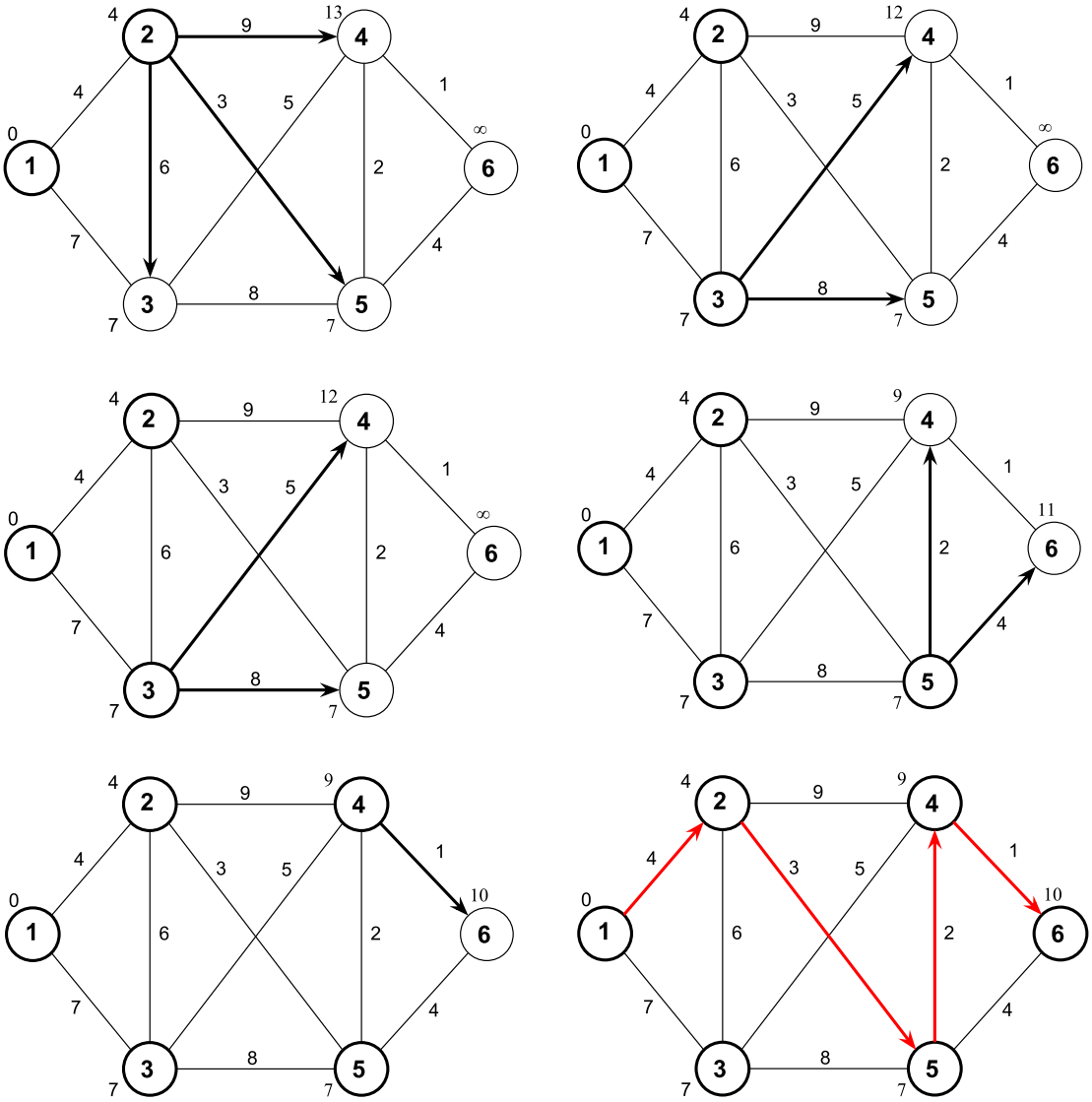
Ako prvé si u všetkých uzlov nastavíme počiatočné vzdialenosti. Jedine uzol 1 bude mať hodnotu dĺžky cesty 0, ostatným uzlom priradíme ∞. Počiatočný uzol je uzol 1, pre jeho následníkov (susedov) aktualizujeme hodnotu $d(v)$.



Obr. 12: Prvý krok riešenia

V ďalšom kroku volíme uzol 2, pretože má najnižšiu hodnotu dĺžky cesty a pre jeho následníkov aktualizujeme vzdialenosti. Zvýraznené uzly považujeme za navštívené.

Takto pokračujeme až kým nenavštívime cieľový uzol 6. Postup riešenia je znázornený na obr. 13.



Obr. 13: Postup riešenia

Výsledkom je najkratšia cesta vedúca cez uzly 1 – 2 – 5 – 4 – 6 s dĺžkou 10.

3.2 Algoritmus A*

Algoritmus A* popísali P. Hart, N. Nilsson a B. Raphael vo svojom článku *A Formal Basis for the Heuristic Determination of Minimum Cost Paths* v roku 1968 [31]. Využíva princípy zhodné s Dijkstrovým algoritmom, ale naviac používa heuristickú funkciu.

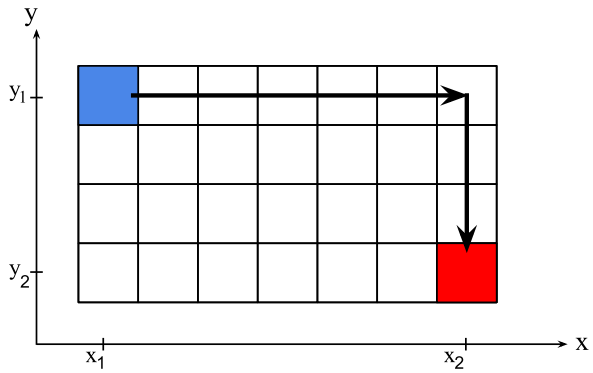
Algoritmus A* je jedným z najznámejších algoritmov pre plánovanie cesty a každá bunka v priestore má priradenú hodnotu

$$f(v) = h(v) + g(v), \tag{2}$$

kde $h(v)$ predstavuje heuristickú vzdialenosť z aktuálneho uzlu do cieľového uzlu, ktorú zvolí užívateľ podľa typu problému a $g(v)$ je dĺžka cesty medzi štartovým a aktuálnym uzlom. Ak pre riešenie problému nebude použitá žiadna heuristika, tzn. $h(v)=0$, dostaneme Dijkstrov algoritmus.

Medzi heuristické funkcie patrí [32]:

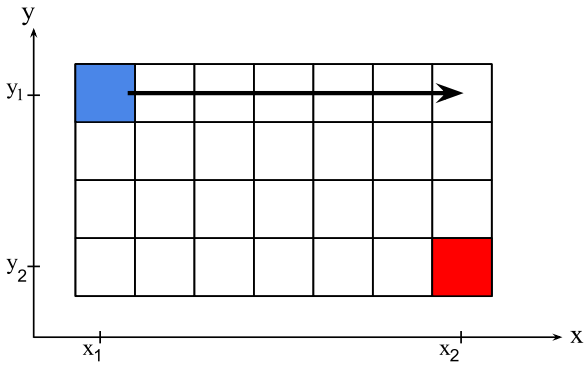
- Manhattanská metrika:
  – mriežka, pohyb v 4 smeroch
  – definovaná vzťahom $d = |x_1 - x_2| + |y_1 - y_2|$, kde d je dĺžka, x_1 a y_1 sú súradnice vybraného uzlu, x_2 a y_2 sú súradnice cieľového uzlu



Obr. 14: Manhattanská metrika

- Diagonálna metrika
  – mriežka, pohyb v 8 smeroch
  – rozlišujeme Čebyševovu vzdialenosť a vzdialenosť *octile*
  – Čebyševova metrika je definovaná vzťahom

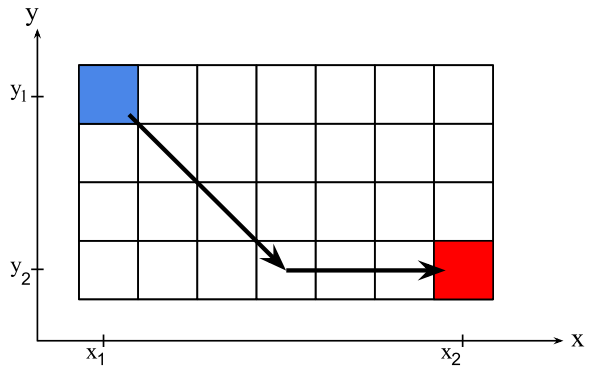
$$d = \max(|x_1 - x_2|, |y_1 - y_2|)$$



Obr. 15: Čebyševova metrika

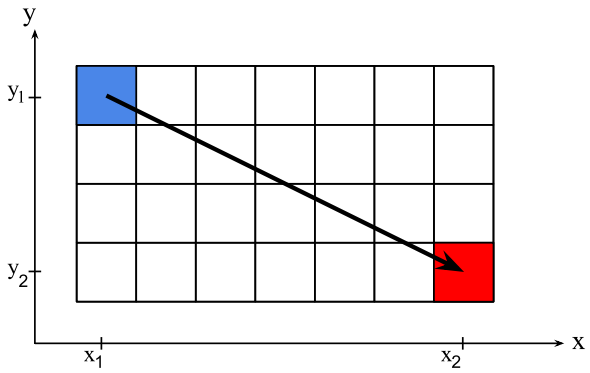
  – metrika octile je definovaná vzťahom

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



Obr. 16: Metrika octile

- Euklidovská metrika:
  – mriežka, akýkoľvek smer pohybu
  – možnosť použiť napr. v grafe viditeľnosti
  – definovaná vzťahom $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$



Obr. 17: Euklidovská metrika

Použitie niektorej z metrík je viazané podmienkou prípustnosti. Použitá heuristická funkcia musí byť prípustná, čo znamená, že nenadhodnocuje vzdialenosť k cieľu. Manhattanská metrika je obmedzená pre mriežku so 4 možnými smermi pohybu. Je nepoužiteľná, ak je priestor reprezentovaný mriežkou s 8 možnými smermi pohybu. V tomto prípade je výsledné riešenie irelevantné.

Ďalej považujeme heuristiku za monotónnu (konzistentnú), ak pre všetky uzly *x,y* splňuje podmienku

$$h(x) \leq d(x, y) + h(y), \tag{3}$$

kde *d*(*x,y*) je dĺžka hrany medzi nimi. Pri použití monotónnej heuristiky je každý uzol navštívený iba raz. [3]

Algoritmus A* používa prioritnú frontu (zoznam OPEN) a zoznam expandovaných uzlov (CLOSED). Na začiatku sa v zozname OPEN nachádza len štartový uzol. V každom cykle algoritmu je vybraný uzol s najvyššou prioritou, teda ten, ktorý má najmenšiu hodnotu *f*(*v*). Aktuálny uzol je presunutý do zoznamu CLOSED a pre jeho následníkov je spočítaná funkcia *f*(*v*), zároveň sú následníci pridaný do zoznamu

OPEN. Ak sa už tieto uzly v zozname nachádzajú, hodnotiaca funkcia sa aktualizuje len v prípade, že nové hodnoty sú nižšie než pôvodné. Algoritmus končí vyprázdnením prioritnej fronty (cesta neexistuje) alebo vybraním cieľového uzlu k expanzii. Výslednú cestu získame sledovaním spätných ukazateľov od cieľového po štartový uzol. [8]

Podobne ako u Dijkstrova algoritmu, kde časová zložitosť závisela na implementácii prioritnej fronty, časová zložitosť A* závisí aj od použitej heuristike. V najhoršom prípade je počet preskúmaných uzlov exponenciálny vzhľadom k dĺžke riešenia (najkratšia cesta). Pre prípad prehľadávaného priestoru tvoreného stromovou štruktúrou je zložitosť polynomiálna. Existuje len jeden optimálny stav a heuristická funkcia h splňuje podmienku

$$|h(x) - h^*(x)| = O\big(log\ h^*(x)\big), \tag{4}$$

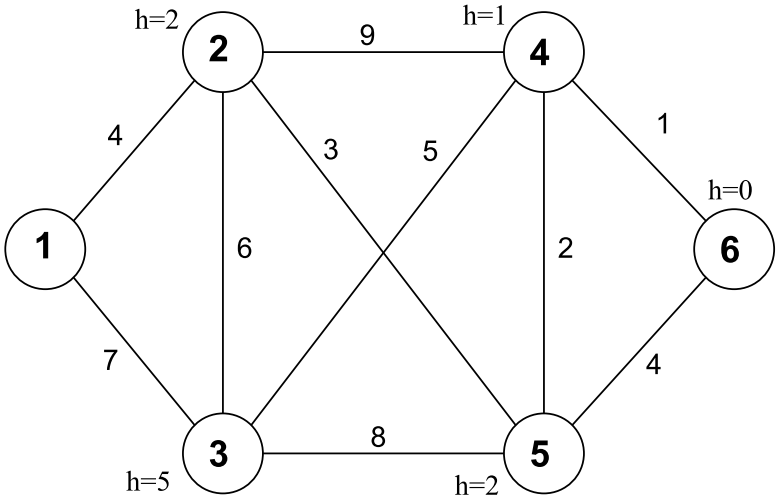
kde h^* je optimálna heuristika, čo znamená presnú vzdialenosť medzi uzlom x a cieľovým uzlom. Inými slovami môžeme povedať, že chyba h nebude rásť rýchlejšie ako logaritmus optimálnej heuristiky. [33]

Algoritmus 2 A* algoritmus

```
A-STAR(G, s, t)
 1  open ← start
 2  closed ← ∅
 3
 4  while open ≠ ∅
 5        u ← EXTRACT-MIN(open)
 6        if u = t break
 7
 8        for each edge e = (u, v)
 9              distance ← g[u] + weight[e]
10
11              if v ∉ open or distance < g[v]
12                    g[v] ← distance
13                    h[v] ← DISTANCE-TO-TARGET(v, t)
14                    f[v] ← g[v] + h[v]
15                    open ← v
16
17        closed ← u
```

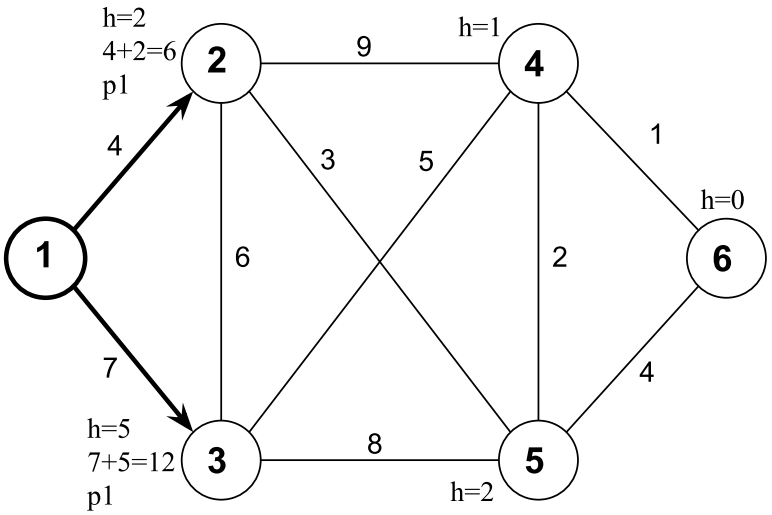
Algoritmus 2

Pre ilustráciu algoritmu A* si uvedieme príklad. Našou úlohou je nájsť cestu v ohodnotenom grafe medzi počiatočným uzlom 1 a cieľovým uzlom 6. V grafe na obr. 18 sú uvedené hodnoty heuristickej funkcie a dĺžky hrán, teda vzdialenosti medzi jednotlivými uzlami.



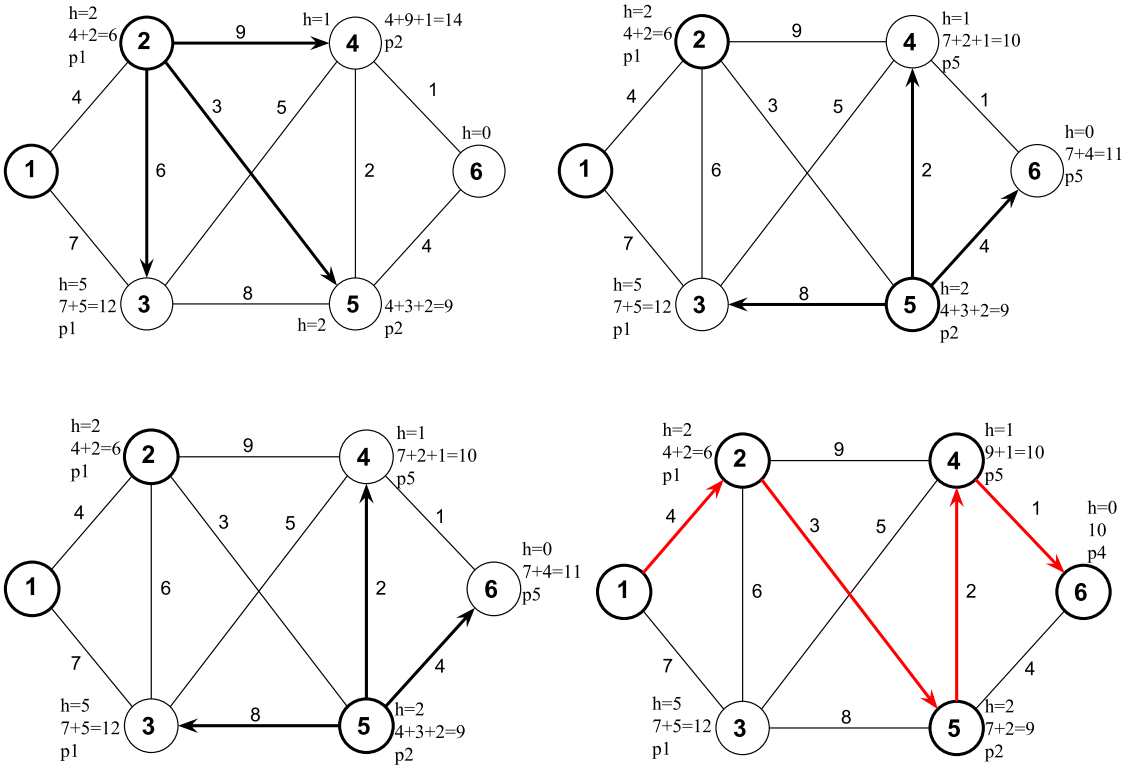
Obr. 18: Graf pre hľadanie cesty algoritmom A*

Na začiatku sa v zozname OPEN nachádza uzol 1, vyberieme ho a presunieme do zoznamu CLOSED a pokračujeme jeho následníkmi, pre ktorých vypočítame hodnotiacu funkciu a vložíme ich do zoznamu OPEN. Vypočítané hodnoty môžeme vidieť v grafe na obrázku. Aby sme vedeli rekonštruovať cestu, k jednotlivým uzlom si zapíšeme predchodcu *p*.



Obr. 19: 1. krok pri riešení hľadania najkratšej cesty algoritmom A*

Z vyššie uvedeného grafu plynie, že budeme expandovať uzol 2. Takto postupujeme kým nie je zoznam OPEN prázdny, alebo kým nevyberieme uzol 6 (cieľový) pre expandovanie. Na obr. 20 je vyznačený postup riešenia a najkratšia cesta vedúca uzlami 1 – 2 – 5 – 4 – 6. Dĺžka cesty je 10.



Obr. 20: Postup riešenia a najkratšia cesta grafom

3.3 Algoritmus NBA*

V súčasnosti sa stretávame so stále väčším množstvom algoritmov pre plánovanie ciest. Vývoj v tomto smere napreduje, prehľadávané sú čoraz väčšie mapy a je žiadané, aby bol čas pre riešenie čo najkratší. Jedným z týchto algoritmov je algoritmus NBA*, ktorý predstavili Wim Pijls a Henk Post vo svojom článku *Yet another bidirectional algorithm for shortest paths* [34].

Ako napovedá názov algoritmu, NBA* je odvodený z už dobre známeho algoritmu A*. Tento algoritmus je označovaný ako obojsmerný, čo znamená, že riešenie je prevádzané z oboch strán súčasne. Pri obojsmernom prehľadávaní grafu je vhodné si pre prehľadnosť kódu určiť proces primárny a sekundárny. Primárny proces je ten, ktorý má počiatok nastavený na štartový uzol prehľadávania, kdežto sekundárny proces ho má nastavený na cieľový uzol prehľadávania. V procese sekundárnom si môžeme použité premenné označiť napr. znakom tilda (~), ako je to uvedené na nasledujúcom pseudokóde.

Algoritmus 3 NBA*

```
NBA(G, s, t)
 1  S ← ∅
 2  R ← ∅
 3  M ← V_G
 4  L ← ∞
 5
 6  for each vertex v ∈ V_G
 7      g[v] = ∞
 8
 9  g[s] ← 0
10  f[s] ← g[s] + h[s]
11
12  while any v∈ V_G has g[v]< ∞
13      u_0 ← EXTRACT-MIN-NODE(g[v] + h[v] | v∈ M)
14      M ← M−{u_0}
15      if g[u_0] + h[u_0] − h[t] ≥ L or g[u_0] + f̃ − h̃[u_0] ≥ L
16          R ← R +{u_0}
17      else
18          S ← S+{u_0}
19          for each edge e = (u_0, v) ∈ E with v∈ M
20              g[v] ← EXTRACT-MIN(g[v], g[u_0] + d[u_0, v])
21              L ← EXTRACT-MIN(L, g[v] + g̃[v])
22      f ← EXTRACT-MIN(g[v] + h[v] | v∈ M)
```

Algoritmus 3

V tomto algoritme rovnako ako u algoritmu A*, $g(v)$ predstavuje ohodnotenie uzlu v. Ďalej je použitý zoznam stabilizovaných uzlov *S*, tj. uzlov, ktorých ohodnotenie sa v priebehu algoritmu nemení a zoznam zamietnutých uzlov *R*, ktoré sa už nebudú expandovať. Zoznam *M* a premenná *L* sú spoločné pre oba prehľadávacie procesy. Premenná *L* obsahuje dĺžku nájdenej cesty zo štartového do cieľového uzlu. Do zoznamu *M* sú na počiatku vložené uzly nachádzajúce sa v grafe medzi štartovým a cieľovým uzlom.

O tom či bude uzol u_0 expandovaný alebo označený ako zamietnutý rozhodujú nasledujúce podmienky:

$$g(u_0) + h(u_0) - h(t) \geq L, \tag{5}$$

$$g(u_0) + \tilde{f} - \tilde{h}(u_0) \geq L \tag{6}$$

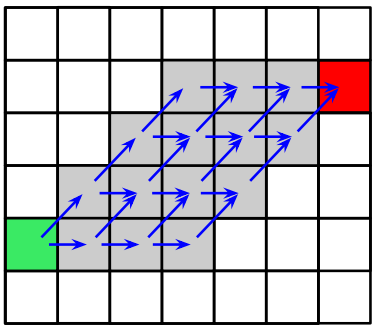
Miesto stretnutia protichodných procesov vznikne v uzle v , ktorý je v oboch procesoch označený ako stabilizovaný. Samotný algoritmus je ukončený v okamihu vyprázdnenia zoznamu *M*. [34; 35]

3.4 Jump Point Search algoritmus (JPS)

Jump point search algoritmus predstavili Daniel Harabor a Alban Grastien vo svojom článku *Online graph pruning for pathfinding on grid maps* [36]. Už z názvu článku plynie, že táto metóda hľadania najkratšej cesty je špecifická pre graf reprezentovaný mriežkou. V tejto kapitole budeme uvažovať mriežku s 8 možnými smermi pohybu, kde horizontálna a vertikálna vzdialenosť medzi bunkami má veľkosť 1, diagonálna $\sqrt{2}$.

JPS je algoritmus kombinujúci A* s jednoduchým operátorom, ktorý expanduje iba určité uzly mriežky, tzv. body skoku „jump points“. Ako vieme, algoritmus A* pridáva do zoznamu ešte neexpandovaných uzlov bezprostredných následníkov aktuálneho uzlu. V priestore reprezentovanom mriežkou sa môžeme stretnúť s možnosťou, že najkratších ciest bude viac. Dĺžka týchto ciest je rovnaká a jediným rozdielom je poradie, v akom sú uzly expandované, tzn. v akom smere sa budeme pohybovať. Príklad viacerých možných ciest je znázornený na obr. 21. Autori vo svojom článku tieto cesty nazývajú symetrické a metóda JPS dokáže eliminovať mnohé z nich. Nasledujúca podkapitola sa podrobne venuje princípu vytvárania bodov skoku.

Autori algoritmu JPS predstavili algoritmus JPS+ [37] ako jeho modifikáciu. Algoritmus JPS+ vyžaduje predspracovanie (pre-processing), ktoré môže trvať až niekoľko milisekúnd. Vzhľadom k dĺžke výpočtu nebude tento algoritmus implementovaný.

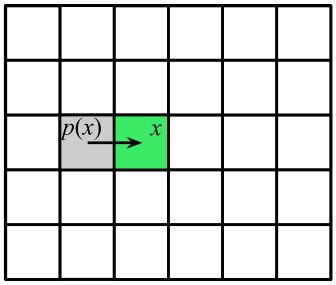


Obr. 21: Mriežka s najkratšími cestami

3.4.1 Body skoku (Jump points)

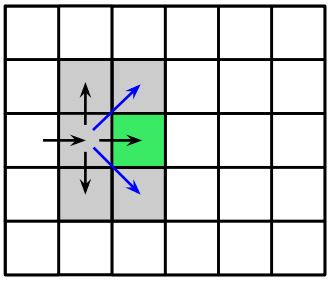
Za účelom rýchlejšieho a optimálneho prehľadávania používa algoritmus JPS stratégiu vytvárania bodov skoku. Základnú myšlienku si znázorníme a popíšeme na základe použitého zdroja [38], z ktorého boli upravené aj nasledujúce obrázky.

Majme uzol x, ktorý expandujeme. O tomto uzle vieme, že jeho predchodca je $p(x)$ a nachádza sa naľavo, teda smer pohybu od $p(x)$ do x je vpravo.



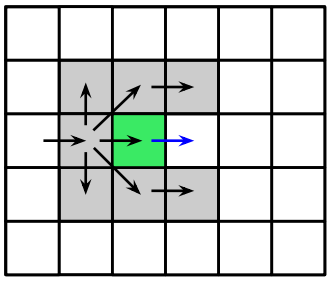
Obr. 22: Smer pohybu vpravo

Všimneme si uzly, ktoré mohli byť navštívené z rodičovského uzlu $p(x)$. Zaujímajú nás susedné uzly, teda uzly nad a pod týmto uzlom a tiež uzly v diagonálnom smere pohybu. Všetky tieto uzly môžu byť z $p(x)$ dosiahnuté kratšou cestou, než aby cesta viedla cez uzol x. Číselne je dĺžka cesty medzi rodičovským uzlom a uzlom na diagonále rovná $\sqrt{2}$, kdežto cez uzol x je dĺžka rovná 2.



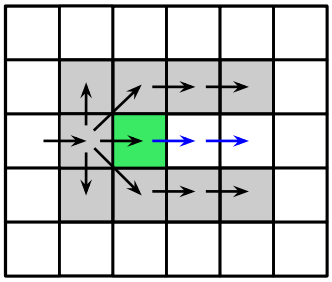
Obr. 23: Susedné uzly $p(x)$

Ďalej si všimneme uzly nachádzajúce sa diagonálne vpravo od uzlu x. Do týchto uzlov sa môžeme dostať cestou priamo z uzlu x, ale tiež zo susedných uzlov nad a pod x. Dĺžka cesty cez uzol x je rovnaká, ale kvôli jednoduchosti budeme predpokladať, že cesta cez susedné uzly je lepšia. Z tohto dôvodu všetky uzly na obrázku označené sivou farbou budeme ignorovať. Ignorovanie uzlov je spojené s prerezávacím pravidlom, ktoré bude vysvetlené v podkapitole 3.4.2.



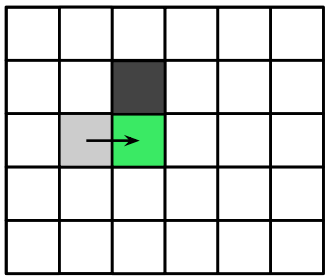
Obr. 24: Ignorované susedné uzly a jediný možný smer z uzlu x

V tomto prípade máme pre expandovanie k dispozícii jediný uzol ležiaci bezprostredne vpravo od uzlu x. Žiaden iný uzol nemôže byť expandovaný, pretože sme predpokladali, že všetci naši susedia sú dosiahnutý prostredníctvom iných (kratších) ciest.



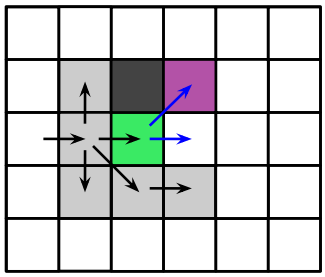
Obr. 25: Pokračovanie v smere pohybu doprava

V smere pohybu napravo a v ignorovaní susedných uzlov pokračujeme až kým nenarazíme na uzol, kde už takto pokračovať nemôžeme. Tento stav môže nastať, ak sa v grafe vyskytuje prekážka nad či pod aktuálnym uzlom, čo môžeme vidieť na obr. 26.



Obr. 26: Výskyt prekážky nad uzlom *x*

Pre tento prípad by sme zopakovali postup a došli by sme do stavu, kde uzol nad nemôže byť dosiahnutý. Ak by prekážka neblokovala cestu, pokračovali by sme expanziou uzlu vpravo ale teraz dochádza k prehodnoteniu situácie. V tejto situácii sme nútený pozrieť sa na uzol, ktorý je napravo od nás a tiež na uzol diagonálne hore. Tohto suseda nazývame nútený sused (*forced neighbor*), pretože ho nemôžeme ignorovať.

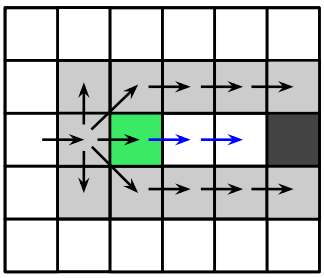


Obr. 27: Nútený sused

V tomto momente je skákanie smerom doprava nemožné a aktuálny uzol je pridaný do zoznamu neexpandovaných uzlov, čo znamená, našli sme bod skoku. Zároveň je tento bod následníkom uzlu *x* a je mu priradená hodnota

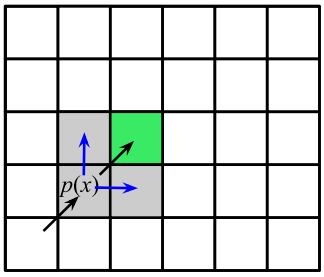
$$g(y) = g(x) + d(x, y), \tag{7}$$

kde *y* je bod skoku, *d*(*x*,*y*) je vzdialenosť medzi uzlami *x* a *y*. Ak pri pohybe doprava dosiahneme prekážku, ďalšie vyhľadávanie v tomto smere je nemožné a nedôjde k vytvoreniu bodu skoku.



Obr. 28: Smer pohybu doprava znemožňuje prekážka

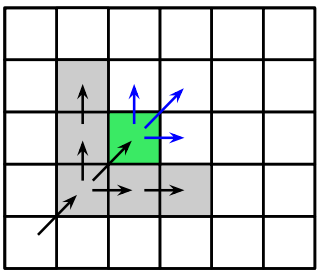
Vysvetlili sme si, ako prebieha generovanie bodov skoku na základe horizontálneho alebo vertikálneho pohybu. Obdobne môžeme body skoku vytvárať diagonálnym pohybom. Pri pohybe po diagonále môžeme aplikovať podobné zjednodušujúce predpoklady ako u horizontálneho či vertikálneho pohybu. V našom prípade sa posunieme diagonálne hore vpravo. Ak sa pozrieme na susedov uzlu x, môžu byť dosiahnuté horizontálnym alebo vertikálnym pohybom z rodičovského uzlu $p(x)$.



Obr. 29: Pohyb diagonálne hore vpravo

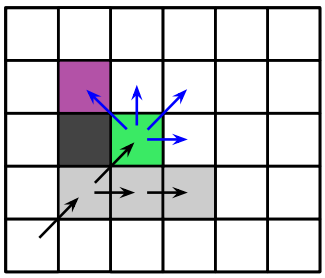
Tiež môžeme predpokladať, že uzly diagonálne vpravo hore a diagonálne vľavo dole od uzlu x môžu byť dosiahnuté prostredníctvom susedov uzla, ktoré sme rozoberali v predchádzajúcom kroku.

Na Obr. 30 vidíme, že musíme zvážiť 3 susedov. Ako prvé sa však zvažujú horizontálne a vertikálne pohyby. V prípade, že žiadny z nich nenájde nový uzol, ktorý by sa pridal do zoznamu OPEN, pokračujeme pohybom po diagonále.



Obr. 30: Možný smer pohybu z uzlu x

Ak sa v priestore vyskytne prekážka naľavo od uzlu x, susedný uzol diagonálne hore vľavo nie je možné dosiahnuť inak než prechodom cez uzol x, čo znamená, že tento uzol je núteným susedom. Tento problém je znázornený na obr. 31. Podobne to platí, ak by bola prekážka umiestnená pod uzlom x. Núteným susedom by bol uzol diagonálne vpravo dole od uzlu x.



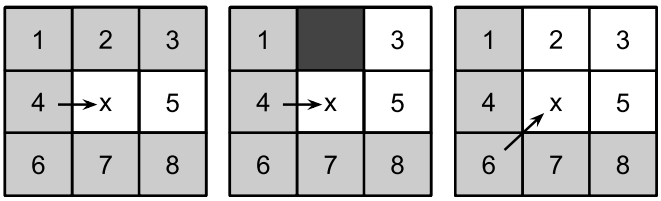
Obr. 31: Nútený sused diagonálne vľavo hore od uzlu *x*

3.4.2 Prerezávacie pravidlá

Pri vytváraní bodov skoku v podkapitole 3.4.1 sme zohľadňovali všetky susedné uzly rodičovského uzlu *p*(*x*). Susedné uzly označíme *n* a prerezávacie pravidlá určujú, či tieto uzly môže byť odrezané (ignorované). Uzol je ignorovaný ak platí jedno z nasledujúcich pravidiel [39]:

- Existuje cesta $\pi' = \langle p, y, n \rangle$ alebo $\pi' = \langle p, n \rangle$, ktorá je jednoznačne kratšia ako cesta $\pi = \langle p, x, n \rangle$, kde *y* je susedný uzol *p* odlišný od *x*
- Existuje cesta $\pi' = \langle p, y, n \rangle$ s rovnakou dĺžkou ako cesta $\pi = \langle p, x, n \rangle$, ale π' má diagonálny pohyb skôr než π

Pre testovanie každého pravidla si všímame susedné uzly aktuálne uzlu *x*. Na obr.32 vidíme, že po aplikovaní týchto pravidiel je možné navštíviť iba susedné uzly označené bielou farbou. Na prostrednom obrázku je cesta cez uzol *y*=2 znemožnená prekážkou. Susedný uzol *n*=3 je označený ako nútený.



Obr. 32: Ignorovanie uzlov

3.4.3 Popis algoritmu

Hľadanie najkratšej cesty v mriežke s využitím algoritmu JPS prebieha tak, že sa spojí prehľadávanie vo vertikálnom, horizontálnom a diagonálnom smere. Pre určenie možného pohybu použijeme rodičovský uzol a snažíme sa v danom smere „skákať“ čo najďalej, čím sa vytvárajú body skoku. Proces, ktorým prebieha identifikovanie bodov skoku označujeme ako *Identify successors*. V ňom uvažujeme len tých susedov aktuálneho uzlu, ktorý neboli odrezaný. Zapísanie každého susedného uzlu do zoznamu OPEN je nežiadúce, preto sa zavolá funkcia *Jump*. Ak je bod skoku nájdený, pridáme ho zoznamu, v opačnom prípade nepridáme nič a pokračujeme, kým nie je množina dostupných susedov vyprázdnená.

Algoritmus 4 Identify successors

```
SUCCESSORS(x:current node, s, t)
1  successors(x) ← ∅
2  neighbours(x) ← PRUNE(x, neighbours(x))
3
4  for each node n ∈ neighbours(x)
5        n ← JUMP(x, direction(x, n), s, t)
6        successors(x) ← n
7  return successors(x)
```

Algoritmus 4

Funkcia *Jump* identifikuje jednotlivých následníkov bodov skoku. Princípom je vyhodnotenie, či má uzol *x* v danom smere následníka *n*. V prípade, že následník existuje, musí overiť, či tento uzol splňuje jednu z nasledujúcich podmienok:

- uzol *y* je cieľovým uzlom,
- uzol *y* má aspoň jedného núteného suseda,
- smer pohybu je diagonálny.

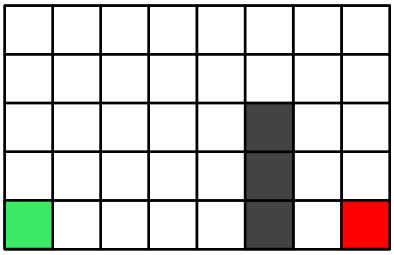
Ak je jedna z podmienok pre uzol *n* splnená, vytvorí sa bod skoku a funkcia tento bod vráti. Funkcia je rekurzívna, ak sa bod skoku nevytvorí. Smer pohybu je zachovaný, ale ako inicializačný uzol je nastavený uzol *n*. Rekurzia končí, ak bude novým uzlom prekážka a nebude možné previesť ďalší krok. Pre zachovanie optimality je nutné, aby algoritmus overil, že v horizontálnom alebo vertikálnom smere sa nenachádza žiadny bod skoku.

Algoritmus 5 Jump

```
JUMP(x:initial node, d⃗:direction, s, t)
 1  n ← STEP(x, d⃗)
 2
 3  if n = obstacle or n = outside
 4        return None
 5  if n = g
 6        return n
 7  if ∃ forced neighbour n' ∈ neighbours(n)
 8        return n
 9  if d⃗ is diagonal
10        for each i ∈ 1, 2
11              if JUMP(n,d⃗ᵢ,s,g) ≠ None
12                    return n
13  return JUMP(n,d⃗,s,g)
```

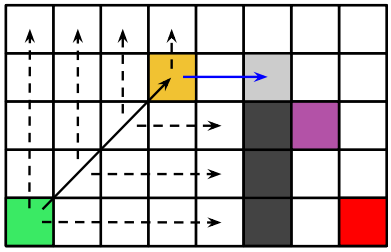
Algoritmus 5

Ak nájdeme bod skoku (viď 3.4.1), pridáme ho do zoznamu OPEN. Každý uzol v zozname je expandovaný v smere závislom na jeho rodičovskom uzle. Na obr. 33 je znázornená mapa a chceme nájsť cestu z počiatočného do cieľového uzla.



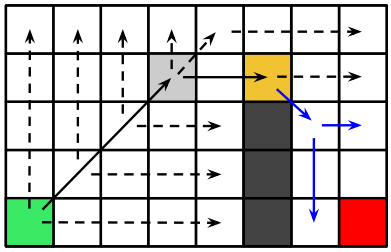
Obr. 33: Mapa pre algoritmus JPS

Ako prvé začíname skúmať horizontálny a vertikálny smer, ak je prehľadávanie ukončené prekážkou, pokračujeme v diagonálnom smere. Na uvedenom príklade vidíme, že prehľadávanie bude pokračovať v diagonálnom smere. V tomto smere pokračujeme celkovo 3 krát. Ďalším krokom je prehľadávanie v horizontálnom smere. Narazíme na núteného suseda a uzol, z ktorého sme sa k nemu dostali zapíšeme do zoznamu OPEN. V zozname OPEN sa tak bude nachádzať uzol sivý, pretože má núteného suseda ale aj uzol žltý. Na obr. 34 je uzol zo zoznamu označený žltou farbou a nútený sused fialovou.



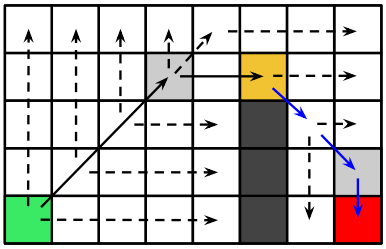
Obr. 34: Smery prehľadávanie

Po preskúmaní horizontálneho a vertikálneho smeru pokračujeme zo žltého uzlu diagonálne, pretože sú preskúmané všetky ostatné smery z tohto uzlu. Ak nenájdeme nič, expandujeme uzol v zozname OPEN, podľa obrázku 34 uzol žltý. Ako prvý prehľadávame horizontálny smer a pretože sa na diagonále nachádza nútený sused, pokračujeme týmto smerom.



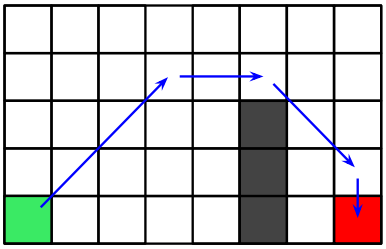
Obr. 35: Pokračovanie v pohybe po diagonále

Z obr. 35 plynie, že ani horizontálne, ani vertikálne nenájdeme nič a opäť pokračujeme diagonálne.



Obr. 36: Bod skoku pri opakovanom diagonálnom pohybe

V uzle označenom sivou farbou pri vertikálnom smere narazíme na cieľový uzol, čo v podstate považujeme za núteného suseda a tento bod je zapísaný do zoznamu OPEN. V poslednom kroku sa tento bod expanduje a algoritmus končí dosiahnutím cieľového uzlu.



Obr. 37: Nájdená cesta algoritmom JPS

4 POPIS APLIKÁCIE

Simulačné prostredie robota je vytvorené v programovacom jazyku Python, čo je vysokoúrovňový skriptovací jazyk. Výber tohto jazyka pre implementáciu bol ovplyvnený možnosťou objektovo orientovaného programovania a knižnicou NetworkX pre vytváranie, manipuláciu a štúdium grafov a sietí. Moje rozhodnutie potvrdila možnosť navrhnutia prehľadného simulačného prostredia prostredníctvom modulu PyGame, ktorý využíva multimediálnu knižnicu SDL. [40; 41; 42]

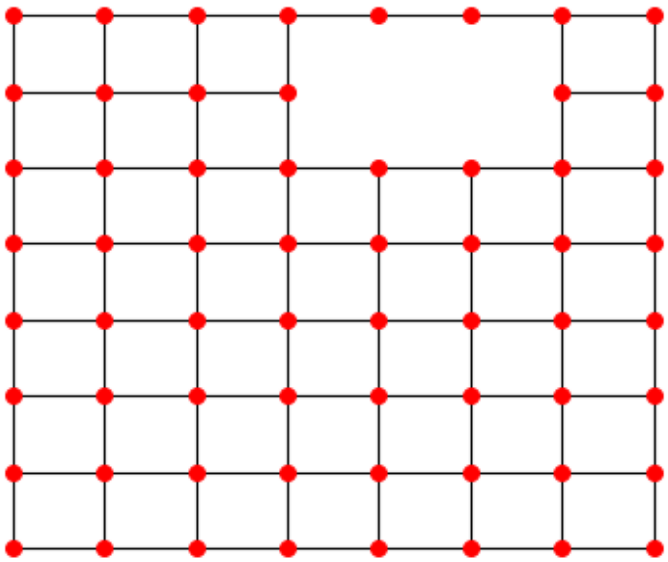
Python

Pre implementáciu vybraných algoritmov a navrhnutie simulačného prostredia bol použitý jazyk Python. Jedná sa o open source a ponúka k dispozícii veľké množstvo knižníc, ktoré programátorovi uľahčujú prácu. Medzi jeho výhody patrí dobrá čitateľnosť kódu a vhodnosť použitia pre začiatočníkov, teda je veľmi jednoduchý z hľadiska učenia.

NetworkX

Ako už bolo spomenuté, NetworkX je knižnica pre jednoduché vytváranie grafov a manipuláciu s nimi. Bez použitia tejto knižnice by bolo nutné vytvoriť si vlastný objekt, ktorý by umožňoval ako pridávanie uzlov a hrán, tak aj ich odstránenie. Táto knižnica umožňuje elegantné vytváranie rôznych grafových štruktúr požadovanej veľkosti a následnú prácu s nimi. V tejto práci je použitým grafom mriežka. Ukážka grafu s prekážkami a kód sú znázonerné na obr. 38.

```
import networkx as nx
G = nx.Graph()
G = nx.grid_2d_graph(8, 8)
G.remove_node((4, 6))
G.remove_node((5, 6))
```



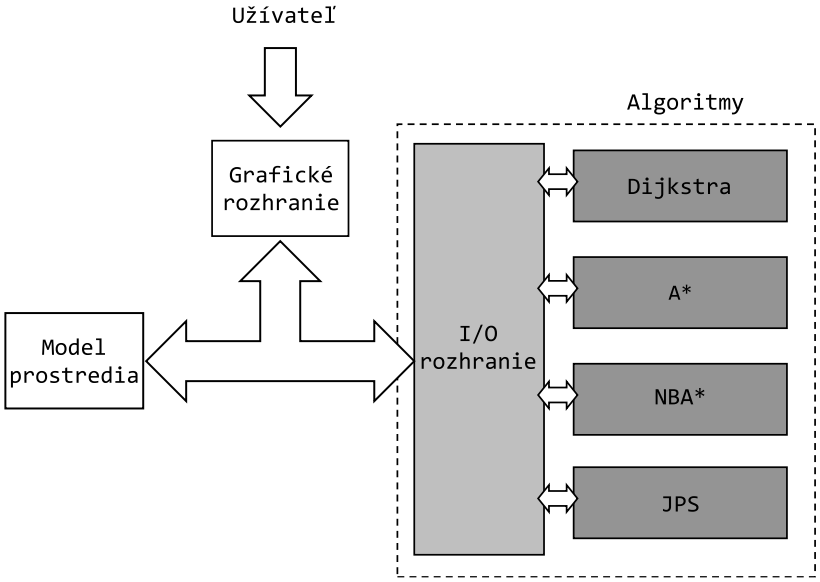
Obr. 38: Ukážka grafu vytvoreného pomocou NetworkX

Pygame

Pygame je bezplatná a voľne dostupná knižnica jazyka Python na vytváranie multimediálnych aplikácií a hier. Táto knižnica umožňuje pracovať s videom, zvukom a vstupnými zariadeniami, pretože je postavená na knižnici SDL (Simple DirectMedia Layer). Pygame rovnako ako SDL pracuje takmer na každej platforme a operačnom systéme.

4.1 Členenie programu

Pri návrhu programu bola zvolená štruktúra programu uvedená na obr. 39.



Obr. 39: Štruktúra programu

Užívateľ pracujúci s týmto programom nemusí používať grafické rozhranie. Program je spustiteľný tiež ako konzolová aplikácia napríklad pomocou skriptu ako je uvedené nižšie:

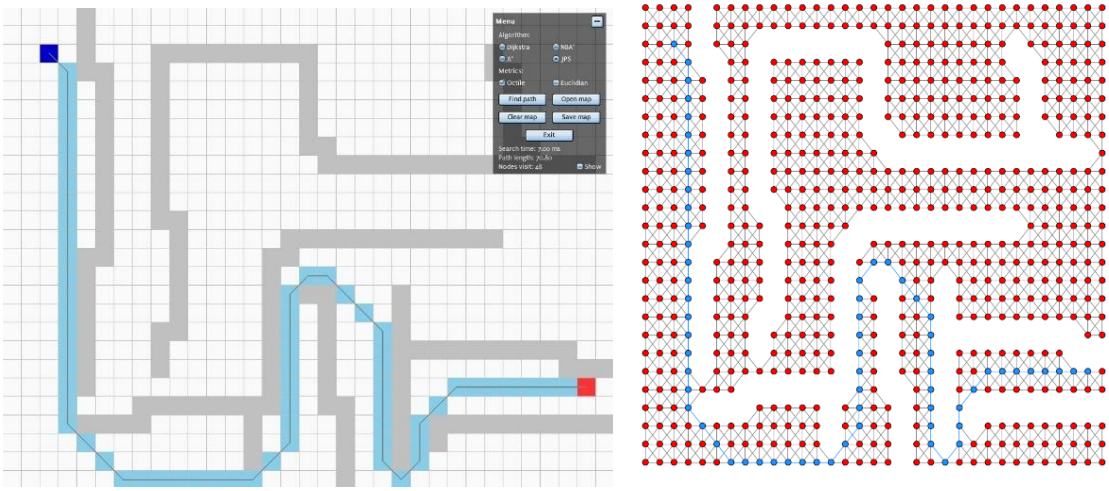
```
nba = NbaStar(G)

tStart = timer()
(pathNba, nbaVisitedNodes) = nba.search(startNode, endNode)
tEnd = timer()

print("NBA*")
print("Search time:    %.3f ms" % (1000 * (tEnd - tStart)))
print("Path length:    %.2f" % nba.getPathLength(pathNba))
print("Nodes explored:", len(nbaVisitedNodes))

print(nbaVisitedNodes)
nba.showGraph(pathNba, nbaVisitedNodes)
```

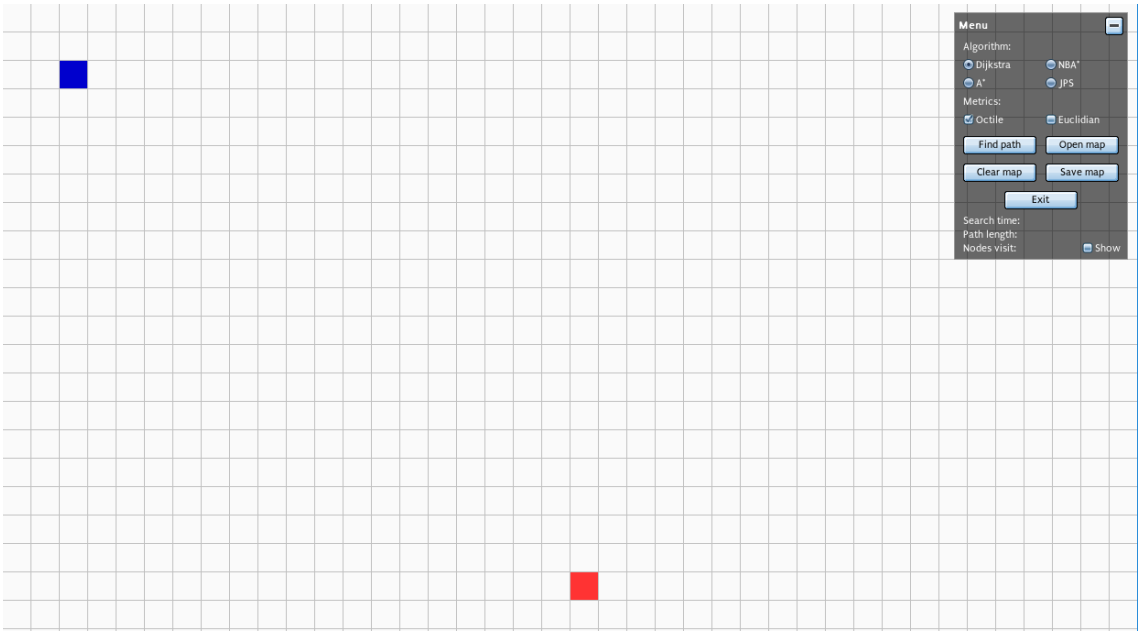
Ak užívateľ zvolí konzolovú aplikáciu, zobrazí sa mu graf vytvorený v NetworkX. Tento graf spolu s vyobrazením najkratšej cesty môžeme vidieť na obr. 40 v porovnaní s grafom vytvoreným pomocou knižnice PyGame.



Obr. 40: Reprezentácia mriežky v knižnici PyGame a NetworkX

4.2 Návrh užívateľského rozhrania

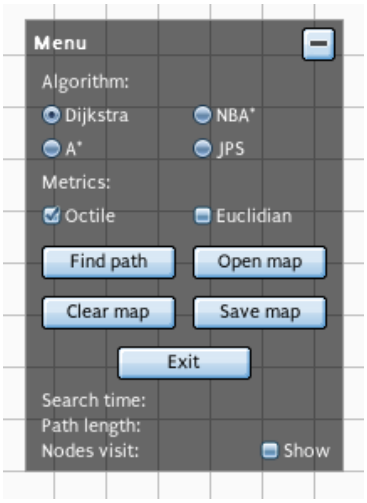
Cieľom návrhu užívateľského rozhrania bolo získať prehľadné prostredie umožňujúce jednoduché ovládanie programu.



Obr. 41: Okno simulačného prostredia

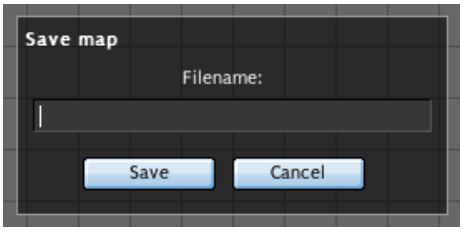
Okno aplikácie, obr.41, môžeme rozdeliť na 2 časti. Jednu časť tvorí hlavné menu, ktoré si môže užívateľ v prípade potreby skryť. Druhú časť predstavuje mapa prostredia, ktorá je pevne daná počtom uzlov použitých v programe. Rozmery použitej mapy sú 40x30 buniek. Veľkosť samotného okna nie je možné meniť.

Na ovládanie programu sa v pravej hornej časti nachádza hlavné menu. Menu obsahuje prepínacie tlačidlá, ktoré umožňujú zvoliť jeden z množiny implementovaných algoritmov. V menu sa tiež nachádzajú tlačidlá pre výber heuristiky, ktorá má byť použitá pre riešenie zadaného problému. Výber heuristiky sa nevzťahuje na Dijkstrov algoritmus, ktorý heuristickú funkciu nevyužíva. Hlavné menu ďalej umožňuje načítať vytvorenú mapu, uložiť novú mapu, prípadne mapu vyčistiť. Vyčistením mapy rozumieme odstránenie existujúcich prekážok, prípadne odstránenie už existujúcej cesty. Zároveň menu obsahuje tlačidlo pre nájdenie cesty, ktoré v mape zvýrazní najkratšiu cestu medzi štartovým a cieľovým bodom a tiež tlačidlo „Exit“, ktoré program ukončí. Pod týmto tlačidlom môžeme nájsť čas potrebný pre nájdenie optimálnej cesty, dĺžku tejto cesty a počet expandovaných uzlov. V menu sa nachádza zaškrtávacie tlačidlo „Show“, ktoré zobrazí expandované uzly. Toto tlačidlo funguje aj po nájdení najkratšej cesty, takže nie je nutné ho zaškrtnúť pred zahájením hľadania cesty.

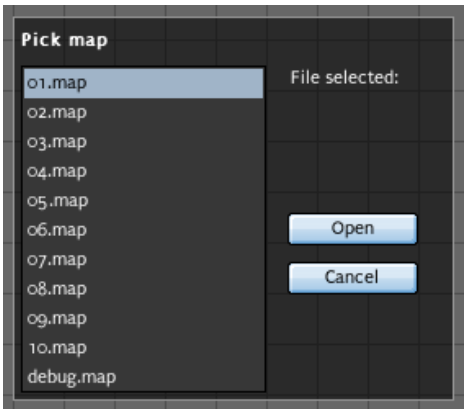


Obr. 42: Hlavné menu aplikácie

Ak užívateľ zvolí možnosť uloženia mapy, zobrazí sa okno, ktoré požiada o názov súboru. Súbor sa uloží do prednastaveného adresára.



Obr. 43: Ukladanie mapy

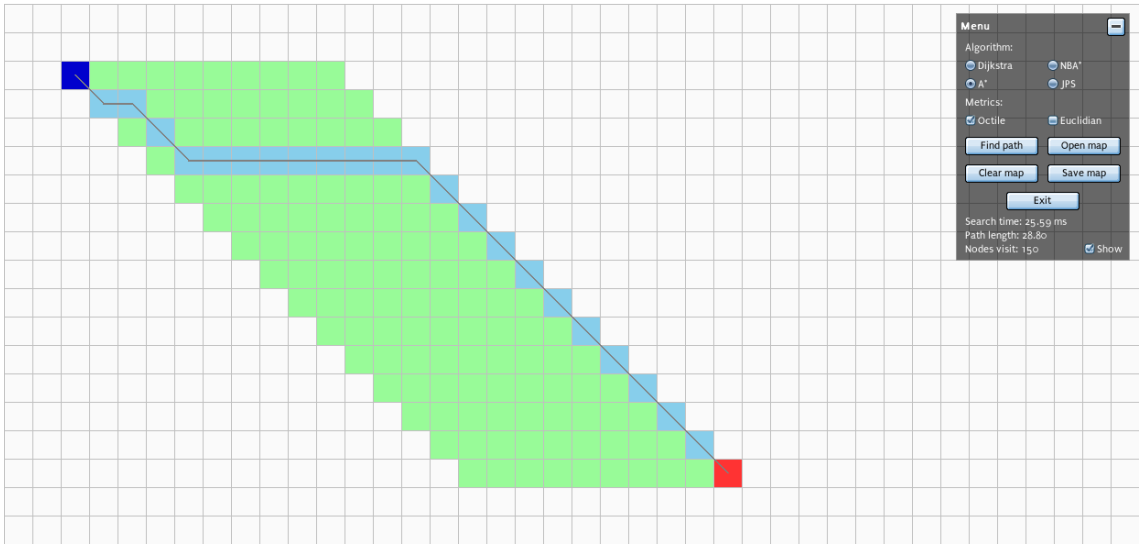


Obr. 44: Výber existujúcej mapy

Po kliknutí na tlačidlo „Open map“ sa zobrazí okno, ktoré dáva užívateľovi možnosť načítať už vytvorenú mapu. Druhou možnosťou je vytvorenie vlastnej mapy, kedy je potrebné nastaviť počiatočnú a cieľovú pozíciu, prípadne prekážky. Nastavenie štartovej a cieľovej pozície je riešené pomocou pravého tlačidla myši. Pre vkladanie prekážok slúži ľavé tlačidlo myši. Ak užívateľ ukončil konfiguráciu pracovného priestoru robota, z hlavného menu si vyberie heuristiku a jeden zo štyroch algoritmov, ktorý chce použiť pre nájdenie najkratšej cesty. Tlačidlom „Find path“ dôjde k zahájeniu riešenia. Nájdená cesta sa v aplikácii zvýrazní.

Pracovná oblasť využíva niekoľko farieb, kde:

- Modrá – reprezentuje štartovaciu pozíciu robota
- Červená – reprezentuje cieľovú pozíciu robota
- Sivá – reprezentuje prekážky
- Svetlá modrá – reprezentuje najkratšiu cestu medzi štartom a cieľom
- Zelená – reprezentuje expandované uzly



Obr. 45: Najkratšia cesta v aplikácii

5 VÝSLEDKY EXPERIMENTOV

Vo vytvorenom simulačnom prostredí boli prevedené dva typy experimentov. Oba experimenty boli realizované na niekoľkých mapách o veľkosti 1200 uzlov, kde dĺžka vertikálnej a horizontálnej hrany bola nastavená na 1 a dĺžka diagonálnej hrany na $\sqrt{2}$. Náročnosť týchto máp sa postupne zvyšovala počtom prekážok. Experimenty prebiehali na počítači s procesorom Intel(R) Core(TM) i7-3770 Cpu @ 3.40Ghz a s operačnou pamäťou RAM 16GB.

V prvom experimente boli jednotlivé algoritmy porovnávané v rámci jednej mapy a to s ohľadom na dĺžku cesty, počet expandovaných uzlov a čas, ktorý bol potrebný pre nájdenie optimálneho riešenia pri použití octile metriky.

V druhom experimente boli použité rovnaké mapy, aby bolo následné porovnanie korektné vzhľadom k prvému experimentu. Jediným rozdielom bola zmena octile metriky na euklidovskú. Na rozdiel od ostatných implementovaných algoritmov, Dijkstrov algoritmus heuristiku nevyužíva a riadi sa iba ohodnotením hrán. Do porovnávacieho experimentu bol použitý aby poukázal na efektivitu informovaných metód.

Prvá mapa

Mapa pre prvé porovnanie bola vytvorená s veľmi nízkou náročnosťou. Do prostredia bola umiestnená len jedna prekážka, ako je možné vidieť na obr. 46 spolu s výsledkami všetkých algoritmov.

V tab. 1 sú uvedené výsledky experimentov pri použití octile heuristiky a v tab.2 pri použití euklidovskej heuristiky.

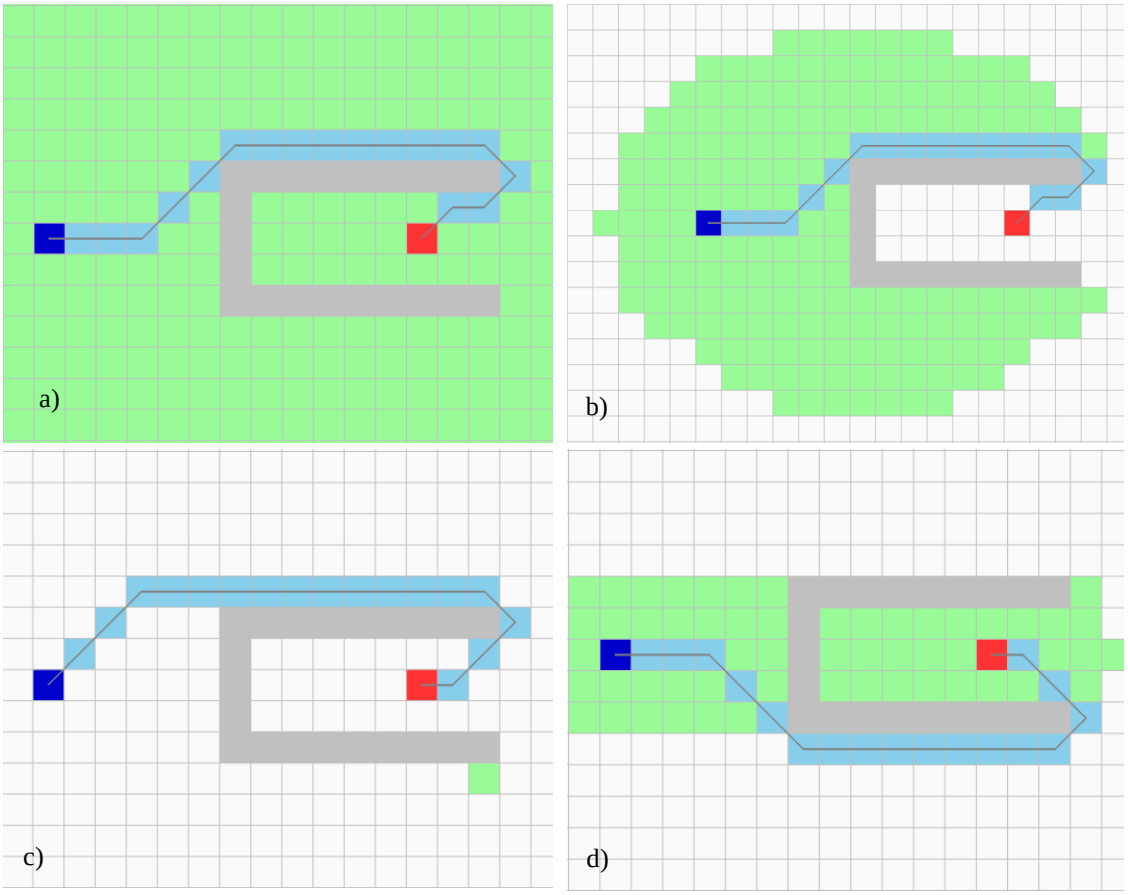
Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	20,49	1265	33,04
A*	20,49	224	10,35
NBA*	20,49	74	12,65
JPS	20,49	5	10,82

Tab. 1: Výsledky prvej mapy, heuristika octile

Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	20,49	1265	37,96
A*	20,49	242	13,96
NBA*	20,49	139	12,33
JPS	20,49	6	11,1

Tab. 2: Výsledky prvej mapy, euklidovská heuristika

Z výsledkov experimentov je vidieť, že pri použité octile heuristiky sú najlepšie výsledky z hľadiska časového dosiahnuté algoritmom A*. Podobne je na tom algoritmus JPS, ktorý výrazne ušetril na počte navštívených uzlov. Ak sa pozrieme na výsledky získané euklidovskou heuristikou, môžeme vidieť že u informovaných algoritmov je počet expandovaných uzlov vyšší a tým sa predĺžil aj čas potrebný pre nájdenie optimálnej cesty.



Obr. 46: Najkratšie cesty v prvej mape získané pomocou: a) Dijkstrovho algoritmu, b) A*, c) JPS, d) NBA* so zobrazením expandovaných uzlov

Druhá mapa

Mapa bola vytvorená pridaním rovnakej prekážky ako v mape prvej, čím sa trochu zvýšila náročnosť. Na obr. 47 sú znázornené najkratšie cesty získané pomocou všetkých algoritmov.

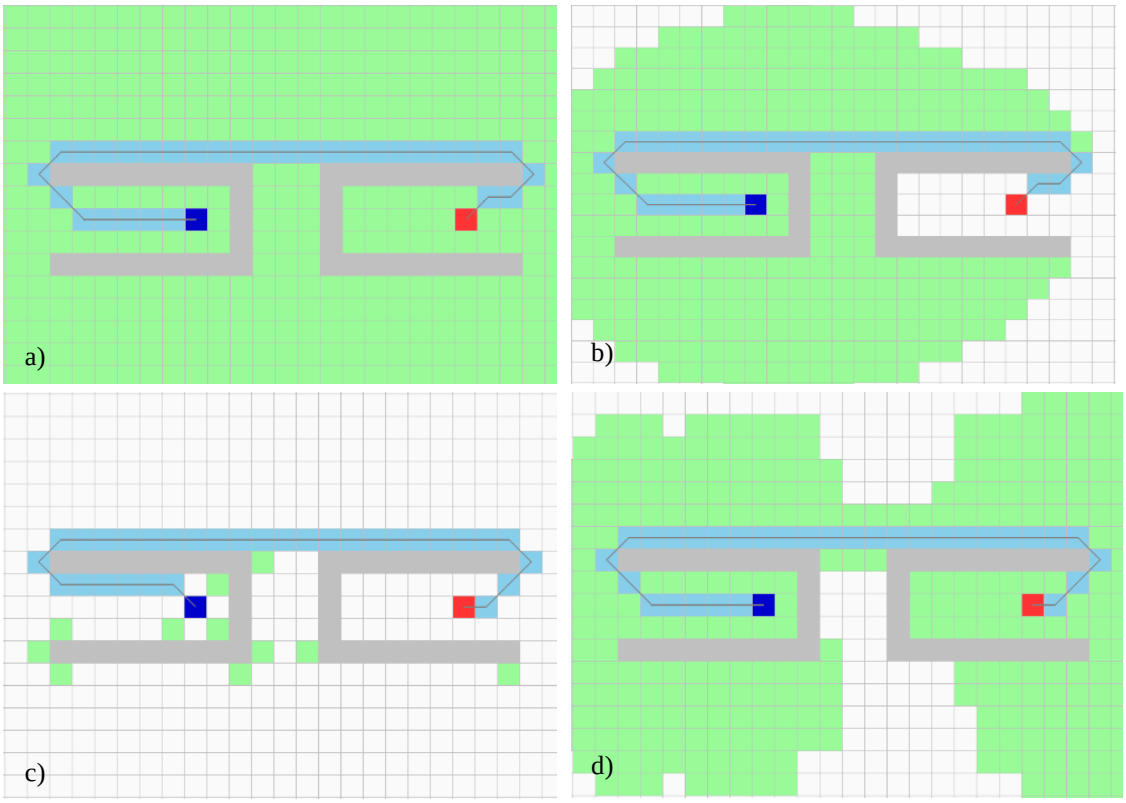
Výsledky experimentov sú zaznamenaná v tabuľkách 3,4. Z výsledkov pre octile heuristiku plynie, že algoritmus JPS dokáže nájsť riešenie v najkratšom čase a pri najmenšom počte expandovaných uzlov. Algoritmus NBA* sa javí voči algoritmu A* ako pomalší, čo je spôsobené ako dopredným tak spätným prehľadávaním. V tejto mape dosiahla heuristika octile jednoznačne lepších výsledkov než euklidovská heuristika.

Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	34,49	1184	39,4
A*	34,49	337	21,28
NBA*	34,49	417	21,74
JPS	34,49	21	10,06

Tab. 3: Výsledky druhej mapy, heuristika octile

Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	34,49	1184	35,54
A*	34,49	385	18,74
NBA*	34,49	511	25,33
JPS	34,49	22	12,94

Tab. 4: Výsledky druhej mapy, euklidovská heuristika



Obr. 47: Najkratšie cesty v druhej mape získané pomocou: a) Dijkstrovho algoritmu, b) A*, c) JPS, d) NBA* so zobrazením expandovaných uzlov

Tretia mapa

Tretia mapa bola vytvorená podľa práce [8]. Táto mapa je stále považovaná za mapu nízkej náročnosti, čo môžeme vidieť na obr. 48.

Zo získaných výsledkov v tab. 5 plynie, že najlepšie si s touto mapou dokázal poradiť algoritmus JPS. Algoritmus A* je časovo porovnateľný, avšak počet expandovaných uzlov je takmer 20krát vyšší. Rovnako ako v predchádzajúcej mape si algoritmus NBA* nedokáže s mapou poradiť rýchlejšie ako A*.

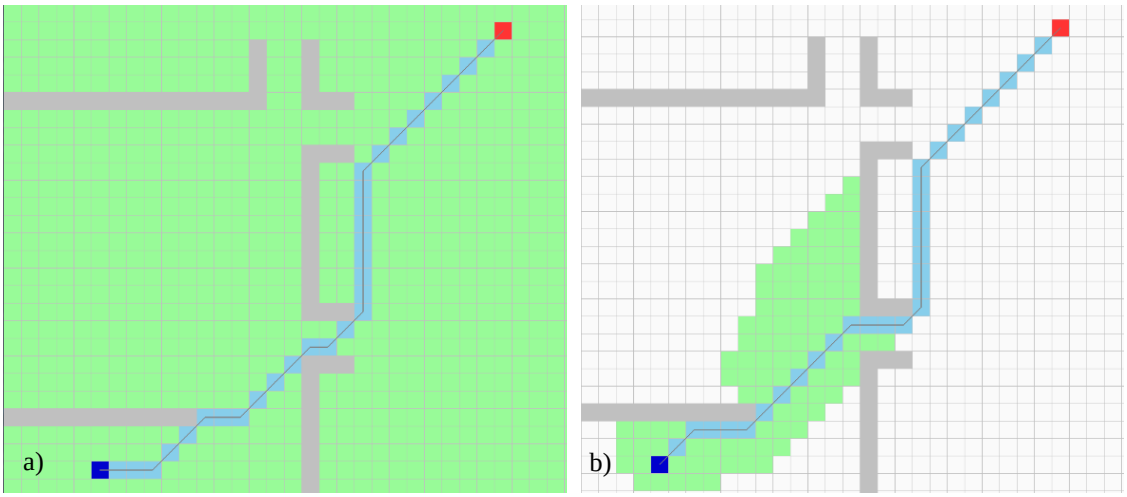
Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	38,04	1152	32,01
A*	38,04	158	8,97
NBA*	38,04	289	19,31
JPS	38,04	8	8,45

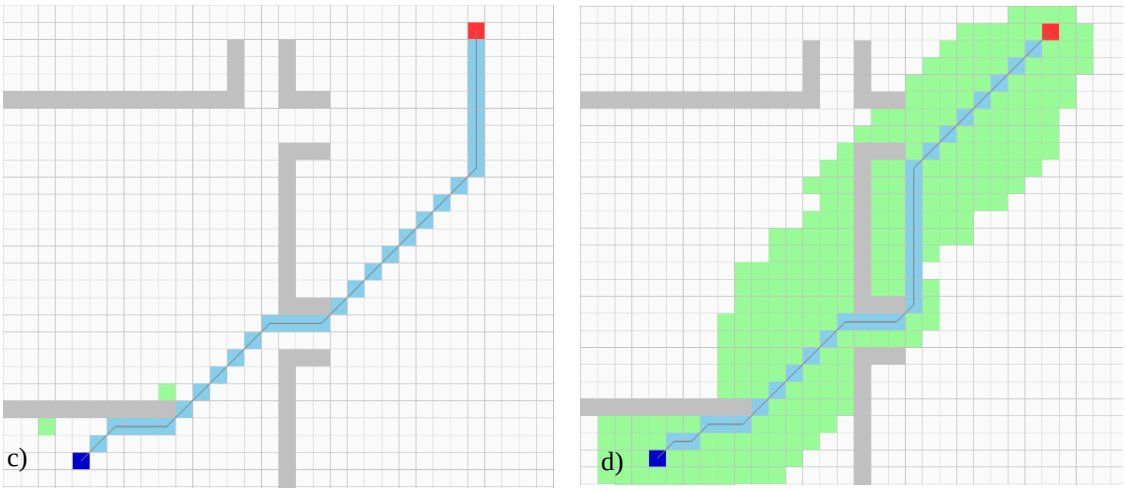
Tab. 5: Výsledky tretej mapy, heuristika octile

V tab. 6 sa nachádzajú výsledky experimentu s použitím euklidovskej heuristiky. Ako najlepší sa jednoznačne ukázal algoritmus JPS.

Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	38,04	1152	33,04
A*	38,04	292	11,92
NBA*	38,04	352	22,64
JPS	38,04	7	9,05

Tab. 6: Výsledky tretej mapy, euklidovská heuristika





Obr. 48: Najkratšie cesty v tretej mape získané pomocou: a) Dijkstrovho algoritmu, b) A*, c) JPS, d) NBA* so zobrazením expandovaných uzlov

Štvrtá mapa

Pre ďalšie porovnanie bola vytvorená mapa s väčším počtom prekážok rozmiestnených po celom priestore, obr. 49.

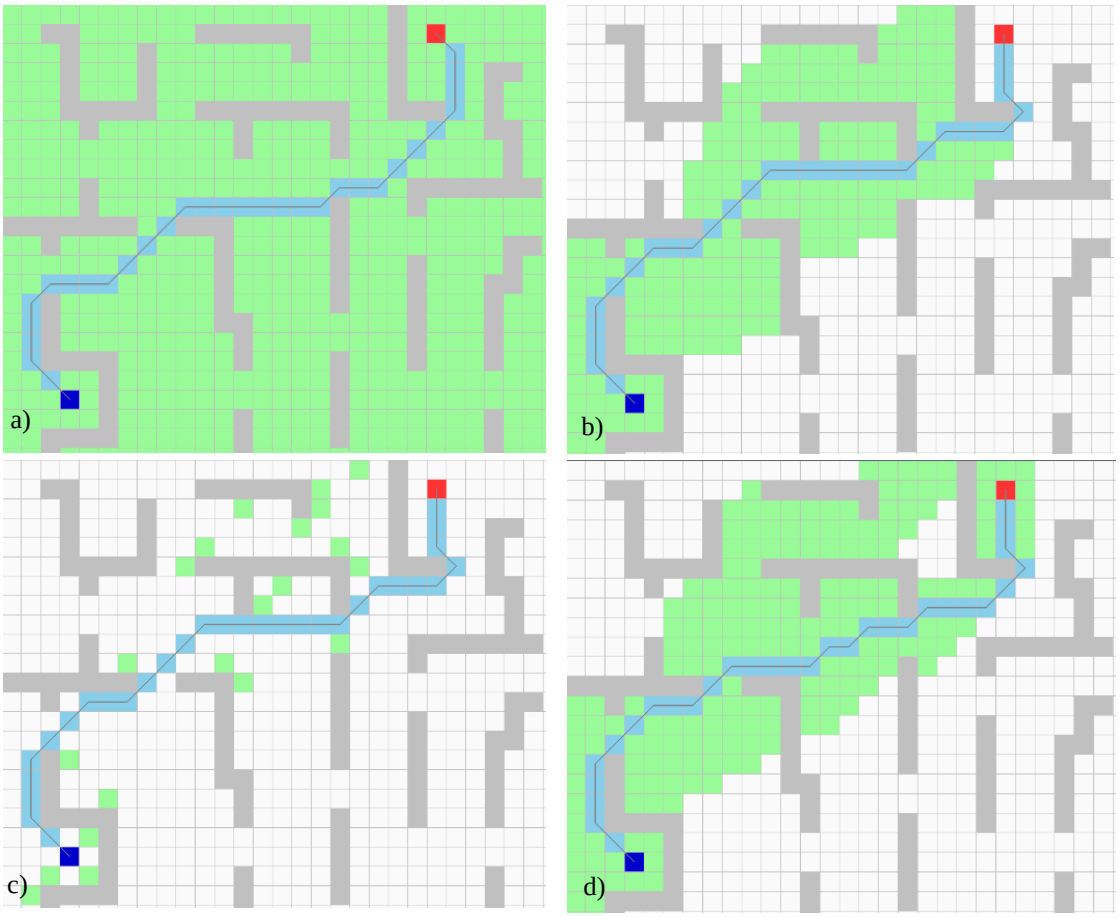
Výsledky experimentov na tejto mape sú zaznamenané v tabuľkách 7,8. Z výsledkov pre octile heuristiku a taktiež pre euklidovskú heuristiku je zrejmé, že algoritmus JPS nájde optimálnu cestu v najkratšom čase. Taktiež si môžeme všimnúť, že pri zložitejších mapách si tento algoritmus vytvára väčší počet bodov skoku, čo ale neovplyvňuje jeho rýchlosť v porovnaní s ostatnými algoritmami. U algoritmu NBA* môžeme opäť vidieť trošku dlhší čas pre nájdenie riešenia, ale počet expandovaných uzlov je nižší než u algoritmu A*.

Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	36,38	1004	24,3
A*	36,38	268	9,41
NBA*	36,38	218	12,82
JPS	36,38	36	7,05

Tab. 7: Výsledky štvrtej mapy, heuristika octile

Algoritmus	Dĺžka	Expandované uzly	Čas [ms]
Dijkstra	36,38	1004	27,44
A*	36,38	292	10,43
NBA*	36,38	301	15,91
JPS	36,38	39	7,18

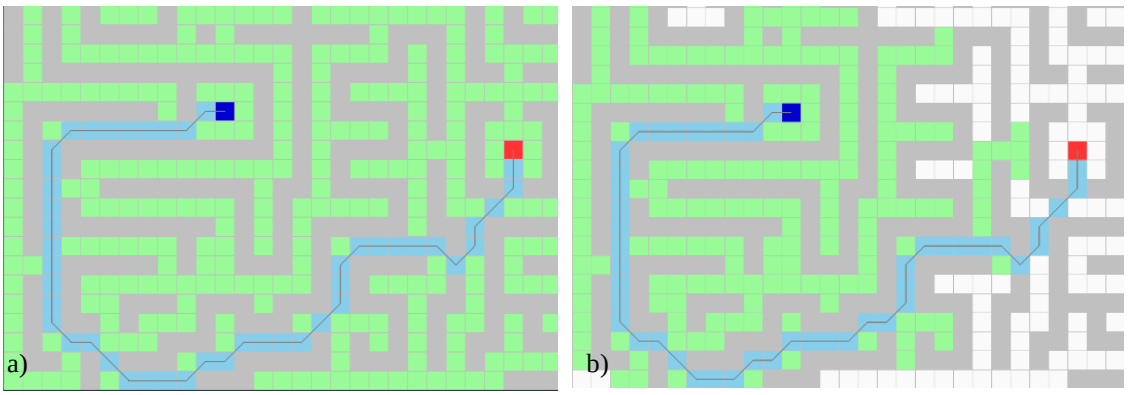
Tab. 8: Výsledky štvrtej mapy, euklidovská heuristika

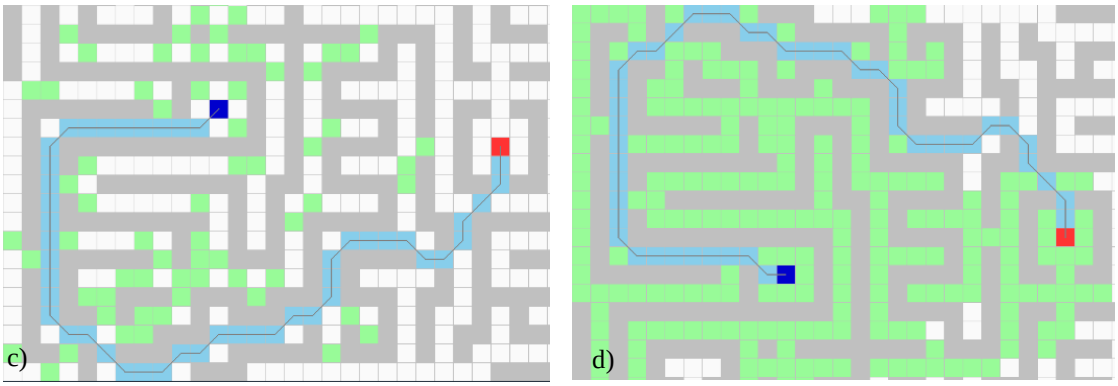


Obr. 49: Najkratšie cesty v štvrtej mape získané pomocou: a) Dijkstrovho algoritmu, b) A*, c) JPS, d) NBA* so zobrazením expandovaných uzlov

Piata mapa

Táto mapa bola vytvorená ako najzložitejšia s veľkým počtom prekážok, kde priestor môžeme považovať za bludisko. Mapa spolu s najkratšou cestou získanou heuristikou octile je znázornená na obr. 50.





Obr. 50: Najkratšie cesty v piatej mape získané pomocou: a) Dijkstrovho algoritmu, b) A*, c) JPS, d) NBA* so zobrazením expandovaných uzlov

Z výsledkov uvedených v tabuľkách 9 a 10 sa ako najlepší algoritmus opäť javí JPS. Dosahuje najlepšie výsledky z hľadiska časovej náročnosti aj počtu expandovaných uzlov pri euklidovskej aj octile heuristike. Zrovnateľný s ním je algoritmus A* čo sa týka časovej náročnosti pri zvýšenom počte expandovaných uzlov. Algoritmus NBA* sa v tejto mape prejavil najhoršie. Ani vďaka použitej octile heuristike nedokázal prekonať Dijkstrov algoritmus, ktorý prehľadal značnú časť mapy. Pri použití euklidovskej heuristiky algoritmus NBA*našiel optimálnu cestu o niečo rýchlejšie než Dijkstrov algoritmus, ale stále je čas riešenia dlhší než u algoritmov A* a JPS.

Algoritmus	**Dĺžka**	**Expandované uzly**	**Čas [ms]**
Dijkstra	52,8	646	10,66
A*	52,8	246	6,98
NBA*	52,8	284	12,7
JPS	52,8	90	6,43

Tab. 9: Výsledky piatej mapy, octile heuristika

Algoritmus	**Dĺžka**	**Expandované uzly**	**Čas [ms]**
Dijkstra	52,8	646	11,06
A*	52,8	255	6,78
NBA*	52,8	289	10,61
JPS	52,8	95	6,66

Tab. 10: Výsledky piatej mapy, euklidovská heuristika

6 ZÁVER

Cieľom tejto diplomovej práce bolo popísať problematiku plánovania cesty mobilného robota. Prvá časť diplomovej práce sa venovala navigácii robota a prístupom k plánovaniu cesty. V ďalšej časti boli popísané vybrané grafové algoritmy pre hľadanie najkratšej cesty v statickom prostredí. Praktická časť sa zaoberala implementáciou algoritmov klasickej umelej inteligencie a návrhom simulačného prostredia.

Pre implementáciu boli vybrané štyri algoritmy (Dijkstra, A*, NBA*, JPS), ktoré pracujú v tomto prostredí, tzn. prekážky sú vopred známe. Navrhnuté simulačné prostredie umožnilo vytvorenie a úpravu máp, ktoré slúžili pre testovanie jednotlivých algoritmov.

V rámci práce boli uskutočnené porovnávacie experimenty na 5 mapách. Náročnosť jednotlivých máp sa postupne zvyšovala pridávaním prekážok. Na týchto mapách bola vyhodnocovaná dĺžka nájdenej cesty, počet expandovaných uzlov a čas potrebný pre nájdenie najkratšej cesty. Vyhodnocovanie prebiehalo na základe použitej heuristiky a to octile a euklidovskej. Z týchto experimentov vyplynulo, že algoritmy založené na heuristickej funkcii nájdu optimálnu cestu rýchlejšie a počet prehľadaných uzlov je nižší, než u algoritmov, ktoré heuristickú funkciu nevyužívajú. Pre porovnanie bol implementovaný Dijktrov algoritmus, ktorý je založený len na ohodnotení hrán. Z realizovaných experimentov ďalej vyplynulo že heuristika octile mala lepší dopad na celkové riešenie problému. Výsledkami experimentu sa potvrdili teoretické znalosti, že pri použití heuristiky octile dôjde k zníženiu počtu expandovaných uzlov. V niektorých prípadoch tvoril výnimku algoritmus NBA*, čo ale záležalo na type použitej mapy. Z implementovaných algoritmov sa na základe experimentov ako najvhodnejší prejavil algoritmus JPS. Zatiaľ čo tento algoritmus pri hľadaní riešenia expandoval najmenší počet uzlov a riešenie našiel v krátkom čase, algoritmus NBA* nesplnil očakávanie pre nájdenie riešenia v statickom prostredí. Ako pri použití heuristiky octile, tak euklidovskej síce expandoval menší počet uzlov než algoritmus A*, ale čas trvania bol dlhší. V budúcnosti by tento problém mohol byť odstránení použitím algoritmu PNBA*, ktorý je paralelnou verziou algoritmu NBA*. Simulačné prostredie umožňuje jednoduché rozšírenie o ďalšie prehľadávacie algoritmy.

7 ZOZNAM POUŽITEJ LITERATÚRY

[1] SARIFF, N. a N. BUNIYAMIN. An Overview of Autonomous Mobile Robot Path Planning Algorithms. In: *2006 4th Student Conference on Research and Development* [online]. IEEE, 2006, s. 183-188 [cit. 2018-05-11]. DOI: 10.1109/SCORED.2006.4339335. ISBN 978-1-4244-0526-8. Dostupné z: http://ieeexplore.ieee.org/document/4339335/

[2] MAC, Thi Thoa, Cosmin COPOT, Duc Trung TRAN a Robin DE KEYSER. Heuristic approaches in robot path planning: A survey. Robotics and Autonomous Systems *[online]. 2016, 86,* 13-28 [cit. 2018-05-25]. DOI: 10.1016/j.robot.2016.08.001. ISSN 09218890. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0921889015300671

[3] DUCHOŇ, František, Andrej BABINEC, Martin KAJAN, Peter BEŇO, Martin FLOREK, Tomáš FICO a Ladislav JURIŠICA. Path Planning with Modified a Star Algorithm for a Mobile Robot. Procedia Engineering [online]. *2014, 96,* 59-69 [cit. 2018-05-11]. DOI: 10.1016/j.proeng.2014.12.098. ISSN 18777058. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S187770581403149X

[4] KRČEK, P. Plánovaní cesty autonomního lokomočního robotu na základě strojového učění. Brno, 2010. Disertace. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. Jiří Dvořák, CSc.

[5] MEYER, Jean-Arcady a David FILLIAT. Map-based navigation in mobile robots. Cognitive Systems Research *[online]. 2003,* 4(4), 283-317 [cit. 2018-05-11]. DOI: 10.1016/S1389-0417(03)00007-X. ISSN 13890417. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S138904170300007X

[6] SEDLÁK, V. Plánování cesty mobilního robotu pomocí mravenčích algoritmů. Brno, 2011. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. Jiří Dvořák, CSc.

[7] ŠINDELÁŘ, J. Plánování cesty neholonomního mobilního robotu. Brno, 2010. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce Ing. Petr Krček.

[8] ČÍŽEK, L. Navigace robotu pomocí grafových algoritmů. Brno, 2011. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. Jiří Dvořák, Csc.

[9] LATOMBE, Jean-Claude. Robot motion planning. 6. print. Boston [u.a.]: Kluwer Acad. Publ, 2001. ISBN 079239206X.

[10] PĚNČÍK, M. Plánování cesty robotu pomocí mravenčích algoritmů. Brno, 2014. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. Jiří Dvořák, CSc.

[11] ŠÍMA, M. Plánovaní cesty robota ve spojitém prostředí. Brno, 2006. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. Jiří Dvořák, CSc.

[12] ŠEDA, M. *Teorie grafů.* VUT FSI v Brně, 2003.

[13] LAVALLE, S.M. Rapidly-exploring random trees: A new tool for path planning [online]. Computer Science Dept., Iowa State University, 1998 [cit. 2018-05-14].

Dostupné z:
ttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.1853&rep=rep1&type=pdf

[14] LAVALLE, S.M. a J. J. KUFFNER. Rapidly-exploring random trees: Progress and Prospects. *Algorithmic and computational robotics: New directions* [online]. 2000, , 293-308 [cit. 2018-05-14]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.1387&rep=rep1&type=pdf

[15] ŠOUSTEK, P. Navigace a spolupráce autonomních robotů založená na metodách umělé inteligence. Brno, 2012. Vysoké učení technické v Brně, Fakulta strojního inženýrství.

[16] ZELINKA, Ivan. SOMA — Self-Organizing Migrating Algorithm. ONWUBOLU, Godfrey *C. a B.* V. BABU. New Optimization Techniques in Engineering [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, s. 167-217 [cit. 2018-05-19]. Studies in Fuzziness and Soft Computing. DOI: 10.1007/978-3-540-39930-8_7. ISBN 978-3-642-05767-0. Dostupné z: http://link.springer.com/10.1007/978-3-540-39930-8_7

[17] YANG, Xin-She. Nature-inspired metaheuristic algorithms. Frome, U.K: Luniver Press, 2008. ISBN 1905986106.

[18] SEDLÁK, V. Plánování cesty mobilního robotu pomocí mravenčích algoritmů. Brno, 2009. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství. Vedoucí práce RNDr. Jiří Dvořák, CSc.

[19] DORIGO, Marco a Thomas STÜTZLE. Ant Colony Optimization: Overview and Recent Advances. GENDREAU, Michel, *ed. a* Jean-Yves POTVIN, ed. Handbook of Metaheuristics [online]. Boston, MA: Springer US, 2010, s. 227-263 [cit. 2018-05-11]. International Series in Operations Research & Management Science. DOI: 10.1007/978-1-4419-1665-5_8. ISBN 978-1-4419-1663-1. Dostupné z: http://link.springer.com/10.1007/978-1-4419-1665-5_8

[20] DREÓ, Johann. Aco branches [online]. b.r. [cit. 2018-05-22]. Dostupné z: https://sk.wikipedia.org/wiki/S%C3%BAbor:Aco_branches.svg

[21] KARABOGA, D. An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report - TR06. Technical Report. *Erciyes* University, Engineering Faculty, Computer Engineering Department, 2005.

[22] CHUROVÁ, V. Optimalizační algoritmy inspirované živou přírodou. Brno, 2015. Bakalářská práce. Masarykova univerzita, Přírodovědecká fakulta. Vedoucí práce Mgr. Et Mgr. Petr Dluhoš.

[23] KENNEDY, J. a R. EBERHART. Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks* [online]. IEEE, 1995, s. 1942-1948 [cit. 2018-05-19]. DOI: 10.1109/ICNN.1995.488968. ISBN 0-7803-2768-3. Dostupné z: http://ieeexplore.ieee.org/document/488968/

[24] STRÁNSKÁ, K. Genetické algoritmy pro řešení úlohy obchodního cestujícího. Olomouc, 2012. Diplomová práce. Univerzita Palackého v Olomouci, Přírodovědecká fakulta. Vedoucí práce RNDr. Pavel Ženčák, Ph.D.

[25] LAMINI, Chaymaa, Said BENHLIMA a Ali ELBEKRI. Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning. Procedia Computer *Science [online]*. 2018, 127, 180-189 [cit. 2018-05-12]. DOI: 10.1016/j.procs.2018.01.113. ISSN 18770509. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S187705091830125X

[26] BŘEZINA, T. a J. DVOŘÁK. *Algoritmy umělé inteligence.* Studijní opora. Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013.

[27] MELUZÍN, P. Lokace zařízení a hledání cest v prostoru. Brno, 2011. Bakalářská práce. Mendelova univerzita v Brně, Provozně ekonomická fakulta. Vedoucí práce Ing. Oldřich Trenz, Ph.D.

[28] STENTZ, A. Optimal and efficient path planning for partially-known environments. In: Proceedings *of the 1994 IEEE International Conference on Robotics and Automation* [online]. IEEE Comput. Soc. Press, 1994, s. 3310-3317 [cit. 2018-05-13]. DOI: 10.1109/ROBOT.1994.351061. ISBN 0-8186-5330-2. Dostupné z: http://ieeexplore.ieee.org/document/351061/

[29] LIKHACHEV, M., D. FERGUSON, G. GORDON, A. STENTZ a S. THRUN. Anytime Dynamic A*: An Anytime, Replanning Algorithm. Proceedings of *the Fifteenth International Conference on International Conference on Automated Planning and Scheduling. AAAI* Press. Monterey, California, USA, 2005, s. 262-271. ISBN 1-57735-220-3.

[30] DIJKSTRA, E. W. A note on two problems in connexion with graphs. Numerische *Mathematik [online]*. 1959, 1(1), 269-271 [cit. 2018-05-13]. DOI: 10.1007/BF01386390. ISSN 0029-599X. Dostupné z: http://link.springer.com/10.1007/BF01386390

[31] HART, Peter, Nils NILSSON a Bertram RAPHAEL. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions *on Systems Science and Cybernetics [online]*. 1968, 4(2), 100-107 [cit. 2018-05-14]. DOI: 10.1109/TSSC.1968.300136. ISSN 0536-1567. Dostupné z: http://ieeexplore.ieee.org/document/4082128/

[32] PATEL, Amit. Heuristics for grid maps [online]. b.r. [cit. 2018-05-14]. Dostupné z: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#heuristics-for-grid-maps

[33] SANTOSO, L.W., A. SETIAWAN a A.K. PRAJOGO. Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path Case Study : Surabaya City Map [online]. Makassar - Indonesia: In: MICEEI 2010, 2010 [cit. 2018-05-14]. Dostupné z: http://fportfolio.petra.ac.id/user_files/03-023/MICEEI2010.pdf

[34] PIJLS, W. a H. POST. Yet another bidirectional algorithm for shortest paths. Report / *Econometric Institute, Erasmus University Rotterdam [online*]. 2009, (2009-10), 1-9 [cit. 2018-05-16]. Dostupné z: http://hdl.handle.net/1765/16100

[35] OLIVEIRA, L.H. a L. CHAIMOWICZ. PNBA*: A Parallel Bidirectional Heuristic Search Algorithm. 2011.

[36] HARABOR, D. a A. GRASTIEN. Online Graph Pruning for Pathfinding on Grid Maps. Proceedings of *the Twenty-Fifth AAAI Conference on Artificial Intelligence. AAAI'11*. San Francisco, California: AAAI Press, 2011, s. 1114-1119.

[37] HARABOR, D. a A. GRASTIEN. Improving Jump Point Search. Proceedings of *the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling. ICAPS'14*. Portsmouth, New Hampshire, USA: AAAI Press, 2014, s. 128-135. ISBN 978-1-57735-660-8.

[38] WITMER, Nathan. Jump Point Search Explained [online]. b.r. [cit. 2018-05-22]. Dostupné z: https://zerowidth.com/2013/05/05/jump-point-search-explained.html

[39] HARABOR, D. a A. GRASTIEN. The JPS Pathfinding system. SOCS. *2012*.

[40] FOUNDATION, PYTHON. Python [online]. b.r. [cit. 2018-05-24]. Dostupné z: https://www.python.org/about/

[41] NETWORKX DEVELOPERS, . NetworkX [online]. b.r. [cit. 2018-05-24]. Dostupné z: https://networkx.github.io/documentation/stable/index.html

[42] AUTOR NEZNÁMY, . About - wiki [online]. b.r. [cit. 2018-05-24]. Dostupné z: https://www.pygame.org/wiki/about