FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Kooperace robotických agentů při sbírání předmětů v bludišti – demonstračních příklady
Student:	Marek Bína
Vedoucí:	Ing. Mgr. Ladislava Smítková Janků, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je navrhnout sérii příkladů postavených na problému kooperace agentů v bludišti při sbírání předmětů, kde se příklady budou vzájemně lišit podmínkami definujícími způsoby spolupráce agentů. Další úkol spočívá v návrhu metodiky pro porovnání vhodnosti různých algoritmů pro hledání cest a přiřazování cílů při řešení navržených příkladů.

1. Proveďte rešerši existujících algoritmů pro hledání cest a přiřazování cílů.
2. Navrhněte sérii příkladů postavených na problému kooperace agentů v bludišti při sbírání předmětů. Příklady se budou vzájemně lišit podmínkami definujícími způsoby spolupráce agentů, zašuměním komunikace apod.
3. Vyberte algoritmy a navrhněte metodiku umožňující porovnat výsledky jejich aplikace.
4. Vyberte nástroje pro implementaci a algoritmy implementujte. Můžete využít existující nástroje/knihovny.
5. Proveďte otestování algoritmů nejprve v softwarovém prostředí, a posléze na skutečných robotech v laboratoři.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 24. února 2022 v Praze.

Bakalářská práce

KOOPERACE ROBOTICKÝCH AGENTŮ PŘI SBÍRÁNÍ PŘEDMĚTŮ V BLUDIŠTI – DEMONSTRAČNÍCH PŘÍKLADY

Marek Bína

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Mgr. Ladislava Smítková Janků Ph.D.
11. května 2023

Odkaz na tuto práci: Bína Marek. *Kooperace robotických agentů při sbírání předmětů v bludišti – demonstračních příklady.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Seznam tabulek

Chtěl bych poděkovat své vedoucí, paní Ing. Mgr. Ladislavě Smítkové Janků Ph.D., za její vedení práce a také své rodině, přátelům a skupině Tangus za jejich podporu, kterou mi věnovali během tvorby této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2023 Marek Bína

Abstrakt

Tato práce se zabývá problémem kooperace agentů při hledání a sběru předmětů v neznámém prostředí na mřížkovém grafu. Nejdříve se v práci popíšou současně používané způsoby řešení problému prohledávání a způsoby rozhodování agentů.

Následně se navrhne série typových příkladů a model, který umožní jejich implementaci v simulovaném prostředí. V práci se provedou experimenty pro kombinace zvolených chování a modulů pro agenta ve vytvořených scénářích. Na základě výsledků se určí vlivy zvolených typů agentů.

Klíčová slova multi-agentní systém, prohledávání bludiště, sbírání předmětů, kooperace agentů, mřížkové grafy

Abstract

This thesis addresses the problem of agent cooperation in finding and collecting objects in an unknown environment on a grid graph. First, it describes the currently used methods for dealing with the exploration problem and agents' decision-making patterns.

Afterwards, a series of typical cases and a model that allow their implementation in a simulated environment is proposed. This work presents experiments for combinations of the chosen behaviors and modules for the agent in the scenarios created. Based on the results, the effects of the selected agent types will be determined.

Keywords multi-agent system, maze exploration, collecting items, agent cooperation, grid graphs

Seznam zkratek

MAS	Multi Agent System
BFS	Breadth First Search
NFA	Nearest Frontier Approach
WFD	Wavefront Frontier Detector
RRA	Reverse Resumable A*
PFDL	Pick First Deliver Later

Úvod

Kooperace agentů a multi-agentní systémy jsou významné témata v oblastech vývoje umělé inteligence a robotiky, kde se dotýkají různých kategorií od vývoje nehráčských charakterů do her po návrh chování skupiny autonomních agentů při hledání osob při požárech/nehodách nebo ztracených v rozsáhlých jeskyních.

Při běžném multi-agentním hledání cest se předpokládá, že každý agent má daný cíl, kde proti tomu v hledání předmětů v neznámém prostředí nemá přímo dán konkretní cíl. Tedy se většinou problém redukuje na problém prohledání celého prostředí.

Algoritmy na kooperativní prohledání bludiště existují, ale při jejich modelaci se provádí různé kompromisy. Například je předpoklad, že existuje centrální možnost sdílení informací či plánování cest. Jiné zase předpokládají, že všichni agenti jsou totožní a můžou vykonávat stejné akce. Tyto algoritmy jsou užitečné, ale v případě diverzních skupin agentů nejsou ideální.

V teoretické části se budu zabývat popsáním potřebných základů z existující literaturu s důrazem na řešení multi-agentního prohledávání neznámého prostředí (bludiště) a algoritmů přidělování cílů.

V praktické části na to navážu návrhem příkladů kooperace agentů, modelu na simulaci chování a popisem použitých algoritmů s jejich implementací. Práce je zakončená prováděním experimentů na jednotlivých příkladech a analýzou vhodnosti použitých algoritmů.

Kapitola 1

Cíle práce

Hlavním cílem této práce je návrh příkladů postavených na problému kooperace agentů při sběru předmětů a prohledávání bludiště a porovnání vhodnosti použití různých algoritmů na navržených příkladech.

Práce nejdříve představí analýzu aktuálních algoritmů týkající se multi-agentního prohledávání a přiřazování cílů agentům v neznámém prostředí. Tyto algoritmy budou následně použity jako základ nebo inspiraci při tvoření chování agentů.

Dalším dílčím krokem, je samotný návrh série několika vzájemně odlišných příkladů postavených na problematice kooperace agentů při sbírání předmětů v bludišti. Příklady se budou lišit ve schopnostech agentů, jejich způsobu spolupráce a rolí operací co umí agenti vykonávat.

Následuje výběr použitých algoritmů. Tyto algoritmy musí umožňovat řešení problému na mřížkovém grafu. Zvolené algoritmy budou porovnávány použitím zvolené metodiky.

Pro možnosti porovnání a experimentace bude potřeba vybrané algoritmy implementovat. Zvolený model musí být dostatečně flexibilní, aby se jednotlivé části dali snadno změnit/nakonfigurovat pro různé agenty podle návrhu scénáře. Model umožní nahrávání různých map a typů agentů.

Nakonec zůstává vyhodnocení použitých algoritmů na jednotlivých příkladech. Na zvolených metrikách se vyhodnotí kvalita chování agentů. Ověří se fungování v závislosti na schopnostech a chování agentů, typech použitých agentů.

Kapitola 2

Teoretická východiska

2.1 Multi-agentní systém

MAS Multi-agentní systém je podoblastí distribuované umělé inteligence, kde se více autonomních entit – agentů snaží kooperovat/konkurovat v daném prostření. [1] Multi-agentní systémy mají široké uplatnění v mnoha disciplínách díky výhodám, které jsou s tím spojené. Mezi ně patří:

- Spolehlivost – fungování systému i přes výpadek části agentů
- Škálovatelnost a flexibilita – agenty lze přidávat a odebírat dle potřeby
- Znovupoužitelnost – agenti fungují jako modulární část a lze je jednodušeji nahradit za jiné [2]

V distribuovaném MAS agenti sdílí společné prostředí, s kterým interagují. Zároveň tím tedy ovlivňují prostředí ostatním agentů. Je třeba, aby agenti predikovali akce vykonané ostatními agenty, aby vhodně zvolili svůj následující krok/cíl a vyhnuli se tak možnému chaosu. V této práci se omezíme jen na statické prostředí. V dynamickém případě (měnící se v čase) se ještě přidá problém určení, zda danou změnu způsobil jiný agent nebo prostředí.

Jednotliví agenti mají omezené vnímání okolí a vykonávají tak rozhodnutí na základě neúplné informace. Toto může vést k suboptimálním výsledků v globálním měřítku.

MAS můžeme rozdělit na *homogenní* a *heterogenní*. U homogenních, agenti jsou si sobě totožné a to v cílech, senzorech, stavech a schopnostech. Rozdíl mezi agenty je v tom, kde se nacházejí v prostředí a jaké mají své lokální cíle. U heterogenních se agenti mohou lišit svojí strukturou a jejich cíle mohou dokonce být v rozporu s cíli jiných agentů. [1]

2.2 Agent

Podle [3], se jako agent dá brát cokoliv, co přes své senzory dokáže získat informace o svém prostředí a působit na něj přes jeho aktuátory. Dále se používá termín „autonomní“ na vyjádření toho, že agent jedná převážně na základě svého stavu, vědomostí a vnímání. [2]

2.3 Rozhodování

Aby se agent choval racionálně, je třeba, aby vždy bral v potaz své znalosti o prostředí. V prostředí, kde agent nemá úplnou informaci, musí řešit, kterou informaci má k dispozici a která mu chybí před vykonáním akce. Toto donutí agenta pracovat interaktivně s prostředím a postupně si doplňovat chybějící znalosti a brát v potaz informace od ostatních agentů. [2]

V kontextu kooperace při prohledávání a sbírání předmětů se jednotliví agenti budou rozhodovat o tom, kam se vydají, aby prohledali nebo aby byli schopni provést interakci s předmětem a dosáhnout tak společného cíle.

2.3.1 Stavové stroje

Mezi možné algoritmy rozhodování patří algoritmy založené na stavových automatech. Agent si udržuje svůj stav. Stav má svou přidruženou úlohu, kterou agent bude vykonávat, pokud se v něm bude nacházet. Dále každý stav má seznam pravidel přechodu, a pokud se splní její podmínky, tak agent přejde do jiného stavu a změní svojí činnost na tu pod novým stavem.

Příkladem může být přechod ze stavu prohledávání na nečinný, při příchodu informace od jiného agenta, že cílený vrchol je již prohledaný.

2.3.2 Stromy chování

Stromy chování neboli *behavior trees* jsou jedním z dalších způsobů jak strukturovat přechody mezi akcemi, které agent vykonává. Jedná se o orientovaný strom (acyklický graf), kde listy představují akci, kterou agent má vykonat nebo vyhodnocení podmínky a vnitřní vrcholy představují řídící uzly. [4] Mohou být navržené expertem nebo naučené pomocí algoritmů strojového učení.

Výhody proti stavovým strojům, stromy chování mají ve formě lepší flexibility, modularity, škálovatelnosti a u rozsáhlejších modelů chování i v čitelnosti.

Pro jednoduché návrhy chování jsou, ale stavové stroje vhodnější díky jejich nenáročné a přirozenější implementaci. [5]

2.3.3 Posilující učení

Posilující učení neboli Reinforcement learning. Jedná se o typ strojového učení, kde se agent učí interaktivně v prostředí pomocí pokusu a omylu. Podle toho zda agent vykonává chtěné/nechtěné operace tak dostane odměnu/postih. [6]

V `MAS` se OpenAI podařilo posilující učení úspěšně uplatnit ve své publikaci [7] na klasickou hru na schovávanou[1], měli dvě skupiny agentů, kde jedna se snažila schovat a druhá ji hledala v prostředí. Schovávající agenti navíc mohli uzamknout pohyblivé překážky a zabránit tak tomu, aby je hledající skupina nalezla. Postupným trénováním se schovávající agenti naučili využít této schopnosti a dosáhnout jejich cíle a to, aby nebyli nalezeni.

[1] Hide and Seek

Tímto způsobem se dá za cenu výkonu potřebného na natrénování udělat docela sofistikované rozhodování agentů a na základě jednoduchých schopností vytvořit komplexní strategie. Jak se také během testování ukázalo, je někdy toto chování agentů velmi nepředvídatelné[2] a hůře se určuje proč danou akci agent vykonal.

2.4 Occupancy grid

Occupancy grid neboli mřížka obsazení je datovou strukturou používanou v robotice při prohledávání světa, hledání bezkonfliktních cest nebo lokalizace agenta ve známém prostředí. Jedná se o jako diskrétní mřížku, kde se v každé buňce udržuje hodnota určující přítomnost překážky. Tato hodnota může být binární (pozice ne/obsahuje překážku) nebo číslo z $\langle 0, 1 \rangle$ určující pravděpodobnost prezence překážky. [8]

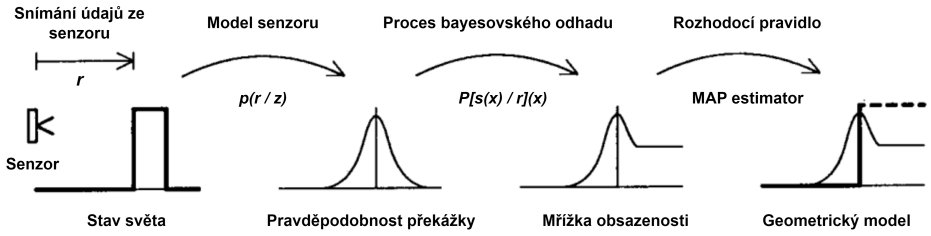
Nebinární mřížka se dá aktualizovat použitím formulí založených na Bayesově vzorci. Předpoklad:

$$Occ(xy) = 1 - Emp(xy) = Map(xy) \tag{2.1}$$

neboli hodnota v mřížce na pozici (x, y) je rovna šanci prezence překážky a doplněk je roven šanci prezence prázdného prostoru. U tohoto způsobu jsou dvě hlavní nevýhody a to, že jedna aktualizace je schopna radikálně změnit hodnotu buňky. A druhá, jakmile buňka dokonvertuje k 0 nebo 1, tak nejde změnit.[9]

$$Map(xy)' = \frac{P_E(xy) * Map(xy)}{P_E(xy) * Map(xy) + (1 - P_E(xy)) * (1 - Map(xy))} \tag{2.2}$$

Mezi další způsoby aktualizace z dat senzorů nebo pro kombinaci map do jedné patří



Obrázek 2.1 Tvoření odhadu mřížky obsazení z údajů senzoru. Převzato z [8]

log-odds, kde se užívá kombinace logaritmu nezávislých pozorování a sčítání. [10]

2.5 Prohledávání prostředí

Problém prohledávání je široce probádaným tématem v rámci robotiky. Prohledávání a mapování patří mezi prerekvizity pro řízení robotů v neznámém prostředí. Cílem těchto procesů je, v co nejkratší čas prozkoumat celé prostředí.

[2] Když hledající agenti shodou náhod nalezli chybu při kolizi, která daného agenta vystřelila do vzduchu a agent tak mohl obejít překážku vytvořenou schovávajícími agenty

Algoritmy se dají rozdělit na dvě skupiny a to:

- reaktivní
- mapově založené – tvoří si vnitřní topologickou reprezentaci prostředí a s ní plánují následující pohyby

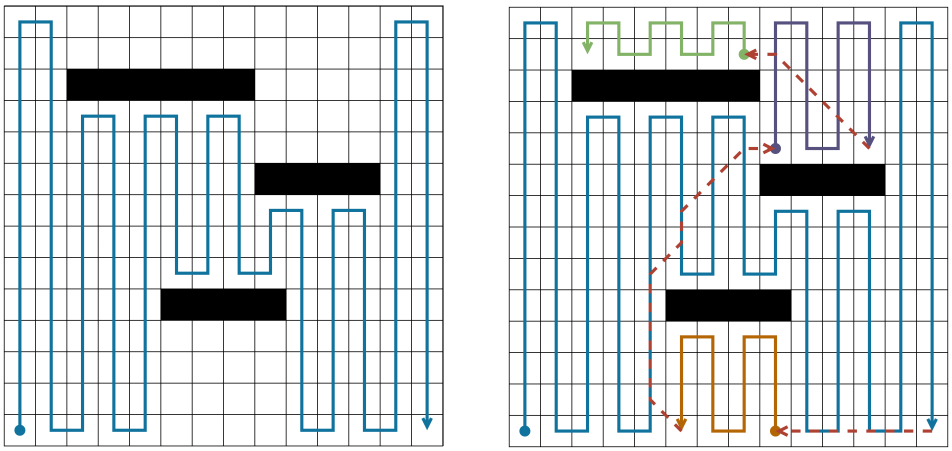
Mezi reaktivní algoritmy patří *Náhodná procházka*, kde v každém kroku algoritmus zvolí náhodný validní směr dokud nenajde cílový vrchol nebo neprohledá celé prostředí. Dále algoritmus *Následování stěny*, který v levé nebo pravé variantě se drží stěny a následuje jí dokud neuvidí cíl. Tento algoritmus často není schopný dosáhnout svého cíle, například v případech nesouvislých ploch. [11]

2.5.1 Boustrophedon pohyb s A*

Mezi jednodušší mapově založené patří *boustrophedon* pohyb, i když se nepoužívá primárně pro prohledávání. Tento pohyb je jednoduchý způsob průchodu tam a zpátky používaný například u domácích vysavačů, které si i díky jen velmi omezeným senzorům (dotykový na detekci nárazu do překážky) dokážou zmapovat neznámé prostředí a v případě potřeby určit nejkratší cestu zpět k nabíjecí stanici přes prozkoumané území.

Algoritmus jednoduše prochází rovně (nezáleží na směru) a při nárazu do překážky se otočí podle nastaveného pořadí směrů, tak aby se vydal směrem, kde je ještě neprojetá buňka. Po přejetí do vedlejší buňky se vydá opačným směrem vůči původnímu.

Na Obrázku 2.2a, lze vidět jak vypadá příklad pohybu. Je vidět, že jeden běh nestačí na průchod všech vrcholů.



(a) Jeden boustrophedon pohyb

(b) Doplnění neprozkoumaných úseků. A* cesty (červené)

Obrázek 2.2 Kompletní pokrytí / prozkoumání pomocí BA*

Proto se tento pohyb kombinuje ještě s A* v BA*[3], kde si robot během svého průchodu ukládá body, které mají neprozkoumaného souseda a použitím algoritmu A* se následně naviguje k nejbližšímu neprozkoumaném bodu. [12] Algoritmus BA* je jedním z algoritmů úplného pokrytí.

[3] BA* – boustrophedon pohyb v kombinaci s A*

Na Obrázku 2.2b je vidět doplnění na kompletní pokrytí prostoru. Čáry různých barev určují jednotlivé běhy *boustrophedon* pohybu, a čárkované červené jsou cesty navržené A* na nejkratší cestu mezi běhy.

2.5.2 Greedy prohledávání

Greedy neboli hladové algoritmy jsou známá technika, kde v každém kroku, algoritmus zvolí lokální optimum. *Greedy* algoritmy tak mohou najít globální optimum, ale v některých případech méně než optimální řešení. [13]

V případě prohledávání prostředí, tento algoritmus jako následující pozici pro agenta zvolí tu nejbližší, co ještě nebyla navštívena / změřena senzory. Výhodou tohoto algoritmu je jednoduchost implementace a integrace do architektury agenta.

Tento způsob má ještě výhodu, že není potřeba, aby po celou dobu mapování, měl plánovač agenta pod kontrolou. Je schopný pokračovat i při změně pozice (přeplánování cesty kvůli potřebě dobít baterii) bez nutnosti vrácení se na pozici, kde skončil.

Dále pokud algoritmus provádí přeplánování po každém kroku, tak se dokáže rychle adaptovat a vždy tak zvolit nejbližší neprozkoumanou pozici nezávisle na tom zda informaci o prostředí dostal ze senzorů nebo přišla od jiných agentů.

Mapování se distribuovat dá mezi více agentů tak, že na každém poběží *Greedy* prohledávání a budou si mezi sebou vzájemně sdílet prozkoumané mapy. [14]

Dalo by se očekávat, že tento algoritmus bude mít celkovou ujetou vzdálenost agentů méně než optimální, ale dle analýzy S. Koenig a C. Tovey [15, 14] lze říci, že i v nejhorších případech je rozdíl v ujetých krocích nevýrazný. Svůj odhad horní hranice dokázali snížit z $O(|V|^{3/2})$ na $O(|V| \ln |V|)$, kde $|V|$ značí počet vrcholů grafu.

K porovnání algoritmus *prohledávání do hloubky* v nejhorším případě projde $2 * |V|$, neboli $O(|V|)$. Do každého vrcholu vstoupí pouze jednou při prvním prohledání a případně ještě jednou při vracení se po doražení do slepé uličky. `DFS` má své nevýhody ve formě nemožnosti pokračovat z nové pozice tedy potřebou se vrátit na poslední pozici, před přerušením. V případě `MAS` má ještě k tomu nemožnost kooperace a nemožnost využití apriorní informace bez větších modifikací. [16]

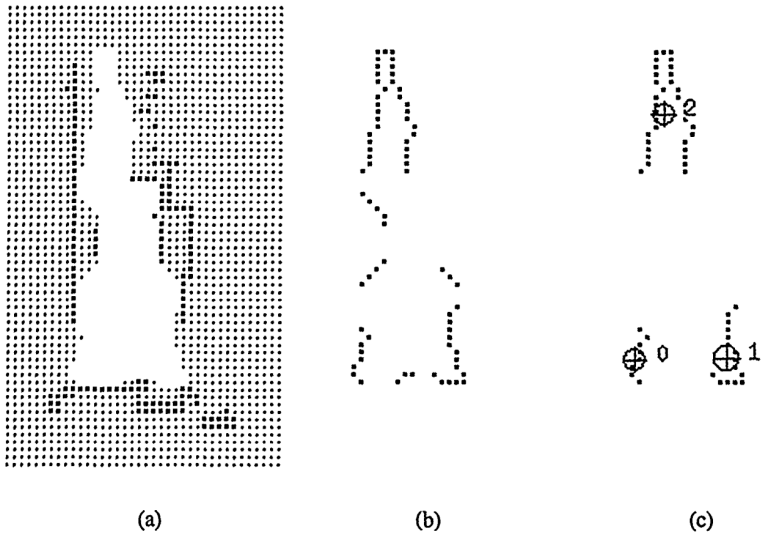
2.5.3 Hraničně založené prohledávání

Frontier-based neboli hraničně založené algoritmy jsou široce využívány pro prohledávání neznámého prostředí. S tímto přístupem přišel Yamauchi v roce 1997. [17] Hranice se zde bere jako mez mezi již prozkoumaným prostorem a neznámým. Sousedící hraniční vrcholy jsou seskupeny do jedné skupiny a podle nastavené minimální velikosti (většinou velikost robota) jsou brány jako jedna hranice.

2.5.3.1 NFA

O rok později Yamauchi přišel s *greedy* algoritmem pro více agentů `NFA` *Nearest Frontier Approach*, kde agenti jednají samostatně, ale sdílejí mezi sebou informace o prostředí. Pro kombinaci jejich map používají *log-odds*. Každý agent si určí existující hranice a jako cílovou si vybere tu nejbližší k jeho poloze. Nevýhodou tohoto přístupu je, že z důvodů samostatné navigace jednotlivých agentů, si agenti mohou zvolit stejnou hranici jako jejich další cíl.

Na Obrázku 2.3, lze vidět detekci hranice z mřížky obsazení a následné vybrání vhodných hraničních bodů na prozkoumání. [10]



Obrázek 2.3 Detekce hranice: (a) Mřížka obsazení, (b) hraniční komponenty, (c) hraniční regiony. Převzato z [10]

Mezi výpočetně nenáročné vylepšení patří opakovaná validace, zda současný cílový vrchol je stále hraničním vrcholem. Tato operace je jen vyhledání v konstantním čase neboli $O(1)$ a dá se jednoduše přidat do původního algoritmu. Dále pro na místnosti dělitelné prostředí se vyplatí přidat preferenci pro prozkoumání hranic v rámci stejného segmentu (místnosti). [18]

Dalšími variantami jsou volba náhodné hranice nebo volba největší. U největší je nevýhoda s potřebou se vracet kvůli neprozkoumaných menších úseků (rohy, menší uličky vedle otevřeného prostoru) a neřešení koordinace mezi agenty.

2.5.3.2 Cena-Utilita

Na vyhnutí se tomu, aby si dva různí agenti zvolili stejný cíl nebo cíl pokrytý jiným agentem, lze využít poměru ceny a utility vrcholu. Cena se bere jako délka nejkratší cesty z pozice agenta ke zvolenému vrcholu a utilita je vyjádřena jako potenciální velikost odhaleného prostředí, který agent pokryje svými senzory při přesunu na danou pozici. [19]

Jeden z algoritmů počítající s utilitou vrcholů lze vidět na Algoritmu 1. Kde na Řádku 6, $P(d)$ vyjadřuje pravděpodobnost, že daný vrchol bude odhalený. Cíl si agent vybírá jako kompromis mezi délkou cesty a potenciálním ziskem z vrcholu.

Algoritmus 1: Cost-Utility [19]

1 Urči se potenciální hraniční vrcholy;
2 Nastav utilitu $U_{x,y}$ pro každý hraniční vrchol na 1;
3 Každý agent i si pro každý vrchol vypočítá délku nejkratší cesty $V^i_{x,y}$;
4 **while** $\exists$ *agent bez cíle* **do**
5 Agent i si zvolí hraniční vrchol splňující $(i, \langle x, y\rangle) = \text{argmax}(U_{x',y'} - V^i_{x',y'})$;
6 Sniž utilitu všech cílových vrcholů v dosahu senzorů podle $U_{x',y'} \leftarrow U_{x',y'} * (1 - P(\|\langle x, y\rangle - \langle x', y'\rangle\|))$

2.5.3.3 Brain-Storm optimalizace

Jedná se o další pokus o vylepšení rozprostření agentů mezi odlišné hranice. Problém prohledávání se zde navíc bere jako optimalizační problém s cílem optimalizovat celkovou ujetou vzdálenost agentů.

Algoritmus navržený podle *G. Li, D. Zhang a Y. Shi* [20], stojí na základech `NFA`, kde si jednotlivé hranice ještě rozdělí na 4 skupiny podle relativní polohy vůči poloze agenta (sever, jih, východ, západ). Jednotliví agenti mají vzdálenostní senzory na způsob laserových dálkových senzorů a komunikují mezi sebou v omezené vzdálenosti. Podobně jako u Yamauchiho implementace `NFA` [10] si agenti při komunikaci sdíleli svojí lokální mapu a jejich pozici.

Pro lepší rozložení agentů se používá brainstorming [21], inspirovaný lidským brainstorming, kde se nejdříve vytvoří seznam potenciálních kandidátů na cílový vrchol. Opakovaně se provádí na základě nastavených pravděpodobností shlukování kandidátů, tvoření nových a nahrazování starých kandidátů za nové/lepší se stejným indexem, dokud neproběhne maximální počet iterací. Při volbě cíle se berou v potaz cílové hranice použité jinými agenty.

2.6 Algoritmus A*

A* je významným algoritmem pro hledání optimální cesty mezi body v grafu. Algoritmus využívá heuristiku odhadující vzdálenost ze současného bodu k cíli na lepší směrování prohledávání.

V každém kroku algoritmus vybere vrchol s nejnižší kombinovanou hodnotou $f(v)$, skutečné vzdálenosti od počátečního vrcholu $g(v)$ a hodnoty heuristické funkce $h(v)$. [22]

$$f(v) = g(v) + h(v) \tag{2.3}$$

V případě mřížkového grafu se heuristika počítá jako manhattanská vzdálenost [23], kde S značí počáteční vrchol a E koncový vrchol.

$$h(S, E) = |(x_S - x_E)| + |(y_S - y_E)| \tag{2.4}$$

Pseudokód A* lze vidět na Algoritmu 2. V případě mřížkového grafu a omezení pohybu čtyřmi směry cena přechodu mezi vrcholy, vyjádřena funkcí *cost* na Řádce 11 bude vždy vracet 1.

Algoritmus 2: A*

```
   Input: start, goal
 1 OPEN ← {(start, h(start))};
 2 CLOSED ← {};
 3 prev ← {(start, start)};
 4 g[start] ← 0;
 5 while OPEN not empty do
 6     node ← OPEN.pop();
 7     CLOSED.add(node);
 8     if node = goal then
 9         return reconstruct_path(goal);
10     for neighbor ∈ neighbors(node)\CLOSED do
11         g′ ← g[node] + cost(node, neighbor);
12         if neighbor ∉ OPEN or g[neighbor] > g′ then
13             g[neighbor] ← g′;
14             prev[neighbor] ← node;
15             f ← g′ + h(neighbor, goal);
16             if neighbor ∈ OPEN then
17                 OPEN.update(neighbor, f);
18             else
19                 OPEN.add((neighbor, f));
```

2.6.1 RRA*

`RRA` *Reverse Resumable A** algoritmus, který popsal Silver [24], kde se algoritmus A* použije v obráceném směru tedy s prohozením vrcholu startu S a cíle C. Jako běžný A* poběží, dokud cílový vrchol (v tomto případě S) nebude expandovaný. Dále pokud použitá heuristika bude konzistentní, pak je dokázáno, že vzdálenost od počátku $g(v)$ bude optimální při expandování vrcholu.

`RRA` se dá použít jako abstraktní heuristika na určení vzdálenosti nejkratší cesty k vrcholu P. Při každém dotazu na nový počáteční vrchol N algoritmus zkontroluje zda v množině uzavřených vrcholů nemá daný vrchol. Pokud ano, tak rovnou vrátí $g(N)$, jinak se pustí `RRA` tam, kde skončil a bude pokračovat, dokud nebude vrchol N expandovaný. Tímto se vyhneme opakovanému propočítávání A* přes společné vrcholy.

Kapitola 3

Návrh a implementace

3.1 Návrh příkladů

V této kapitole je popis příkladů, na základě kterých byly následně určeny ty vlastnosti, které má obsahovat implementovaný model.

- Prvním příkladem na ověření fungování je klasické bludiště s hledáním předmětů, kde cílem je v co nejméně krocích nalézt všechny předměty. U tohoto příkladu se řeší čistě rozdíl ve zvoleném navigačním algoritmu.
- Druhý příklad počítá se dvěma skupinami agentů. Jako případ užití se dají brát `pomocníci` a `dělníci`, kde předměty mohou reprezentovat objekty, které dělníci mají opravit a jejich pomocníci mají urychlit práci dělníkům tím, že budou vyhledávat a následně informovat dělníky ohledně pozic nalezených předmětů.
- V třetím příkladě, bude jako u předchozího předpoklad alespoň 2 skupin agentů. V tomto případě se budou lišit tím, že jedna skupina bude moct předmět zvednout, ale předmět se nebude brát jako hotový/ukončený, dokud předmět nebude předaný cílené skupině agentů. Pod případem užití se dají představit skupiny `sběračů` a `nosičů`, kde sběrač nemá kapacitu na více předmětů a musí před zvednutím dalšího předmětu, předat ten co má u sebe nosiči.
- A jako čtvrtý byl zvolený případ, kde cílem je odnosit předměty z prostředí na předem určené shromaždiště (vynosit předměty z bludiště ven). U tohoto případu se dá řešit chování při nalezení předmětu. Zda předmět vzít rovnou a vrátit se s ním na shromaždiště (pokud je známé) nebo nejdříve prohledat celé prostředí a následně se vrátit pro předměty.

3.2 Model prostředí

Prostředí udržuje objekty potřebné pro běh simulace. Objekty jako mřížková reprezentace mapy, agenti, předměty a případně cíle. Následně umožňuje předání potřebných objektů agentům pro umožnění interakce mezi agent-bludiště, agent-předmět a agent-agent.

Model v každém kroku simulace vytvoří permutaci agentů, na kterých následně postupně vyvolá tři operace. Operace rozdělující akce na `před pohybem`, `pohyb` a `po pohybu`.

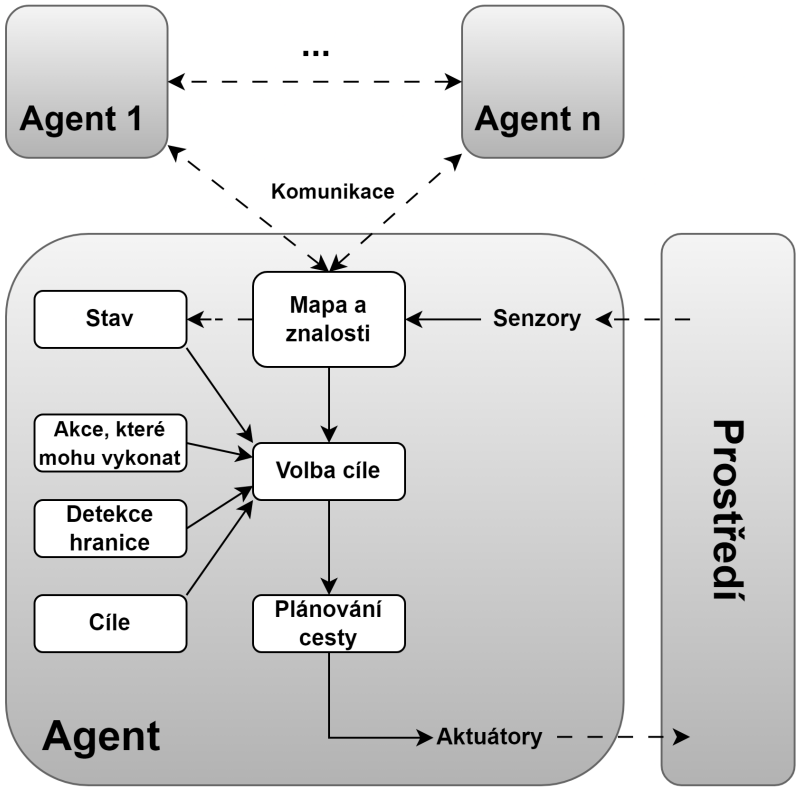
3.3 Návrh agenta

Zvolený návrh agenta je modulární pro možnost volby a modifikace použitých senzorů, způsobu komunikace a pravidel rozhodování agenta při vybírání následujícího cíle a volby vrcholu pro prohledání. Mezi zvolitelné části agenta patří:

- Senzory
- Komunikační modul
- Rozhodovací algoritmus
- Prohledávací algoritmus

Tyto jednotlivé části budou dále popsány podrobněji.

Jedná se o modelem a cílem založeného agenta, který si udržuje stav prostředí a vybírá akce, které vedou k splnění jeho cílů. [3] Návrh agenta lze vidět na Obrázku 3.1.



Obrázek 3.1 Návrh agenta se znalostí prostředí a cíli. Kombinace návrhů z [3, 20]

Agent si udržuje svou lokální reprezentaci prostředí (třída `Knowledge` – vědomosti). Obsahem vědomostí je stav prohledání jednotlivých vrcholů mapy a jejich typ při odhalení, vzpomínky na poslední stavy agentů/předmětů, které byly získané ze senzorů agenta nebo z dat od ostatních agentů. Tuto paměť může za splněných podmínek ke komunikaci sdílet s ostatními agenty v jeho okolí za účelem kooperace a rychlejšího naplnění společných cílů.

Pro navigaci skrze bludiště a vyhýbání se překážkám agent využívá algoritmu A* popsaného v Kapitole 2.6. Algoritmus pracuje s mapou uloženou v paměti agenta a hledá cestu skrze bludiště přes prohledané vrcholy. Je také umožněno, aby navigoval i přes neprozkoumané, toto se může hodit v případě nesouvislých komponent v reprezentaci mapy mezi zvoleným cílem a pozicí agenta. Nevýhodou je nutnost přeplánování v případě objevení překážky v nově prozkoumaných vrcholech po cestě.

Akce, které agent výkoná v rámci operace pro každý krok simulace:

- `pre_step`
  - Agent na základě hodnot ve své paměti, svého současného stavu a cíle / rozhodne následující kroky, přechod mezi stavy
  - Vykoná činnosti, které nejsou krokem
  - Rozešle informace agentům v dosahu
- `step`
  - Agent bude následovat svůj vytvořený plán z předchozího bodu a udělá krok, pokud to není v kolizi se stavem prostředí
- `post_step`
  - Agent si vyčistí zápis posledních změn
  - použije své senzory na vnímání prostředí a výstup uloží do paměti

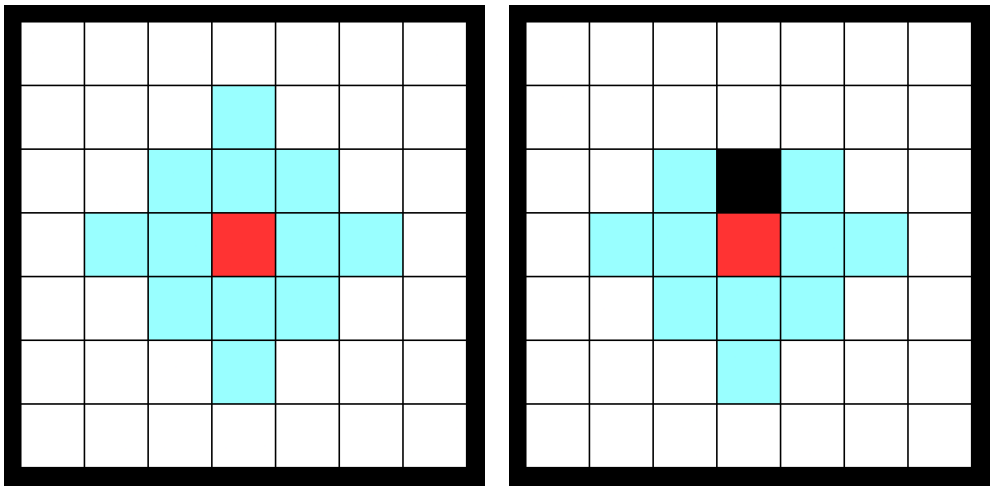
Tímto způsobem je zaručeno rozeslání aktuálních plánů a stavu okolí po interakcích. Dále při zobrazování výstupu simulace je na prozkoumané mapě znázorněn nejnovější stav ze senzorů agentů.

3.3.1 Senzory

Aby agent mohl prohledávat prostředí je potřeba, aby byl vybaven senzory, které mu umožní získat informace o prezenci překážek v jeho okolí či pozici hledaných předmětů. V rámci simulace se omezíme na `senzor sousedů`, který je schopný získat informace o stavu vrcholů v přímém okolí agenta. Dále na, `kruhový senzor`, který reprezentuje 360° vizi okolo agenta v omezené vzdálenosti.

Tento senzor získá informaci o vrcholech, pokud k nim vede přímá nepřerušená linka od středu robota do středu vrcholu délky v limitu dosahu. V rámci programu je umožněno přidat i další senzory, které nejsou takto všestranné, například zorné pole nebo směrový. Přidáním těchto senzorů by vznikla nutnost přidání aktivní rotace při doražení agenta do cílového vrcholu do rozhodování agenta. Tímto krokem by dosáhl stejného výsledku jako při využití všestranného senzoru, ale za cenu většího počtu akcí.

Na Obrázku 3.2, lze vidět porovnání viditelných vrcholů v blokovaném a neblokovaném případě pro kruhový senzor s dosahem 2 bloky. Světlemodrou jsou zobrazeny viditelné vrcholy, červenou agent a černou překážka.



(a) Neblokovaná **(b)** Blokovaná

Obrázek 3.2 Porovnání ne/blokované vize s dosahem 2

V rámci simulace budeme předpokládat, že tento senzor bude vracet stav daného vrcholu s jistotou. Tento předpoklad, avšak neplatí u reálných senzorů, které mají nepřesné měření (šum) a problémy s možnými odrazy od lesklých povrchů. V těchto případech se pracuje s *occupancy grid* neboli mřížkou obsazenosti udržující pravděpodobnost prezence překážky, popsané v Kapitole 2.4.

Bresenhamův linkový algoritmus

Jako optimalizaci pro určení zda daný vrchol je viditelný a snížení počtu dotazů na překážku v mapě, byl zvolen Bresenhamův algoritmus na kreslení čar. Tento algoritmus byl vytvořený za cílem rychle aproximovat/vykreslit linku mezi dvěma body (pixely na obrazovce). [25] Tento algoritmus je rychlý a efektivní na určení vrcholů, kterými linka prochází, v našem případě linka vize agenta a vrcholů, kde se může nacházet překážka, co by omezila vizi agenta.

Pseudokód je vidět viz Algoritmus 3. Místo vykreslování barvy pixelu se na Řádku 7 provádí kontrola zda daná koordináta není mimo mřížku nebo zda se na tomto místě nenachází překážka.

Tento algoritmus se dá celý přeskočit, pokud nebudou splněny podmínky:

- Cílová pozice je validní a na mřížce není označena jako stěna
- Euklidovská vzdálenost [26] bodů je menší než maximální dosah senzoru

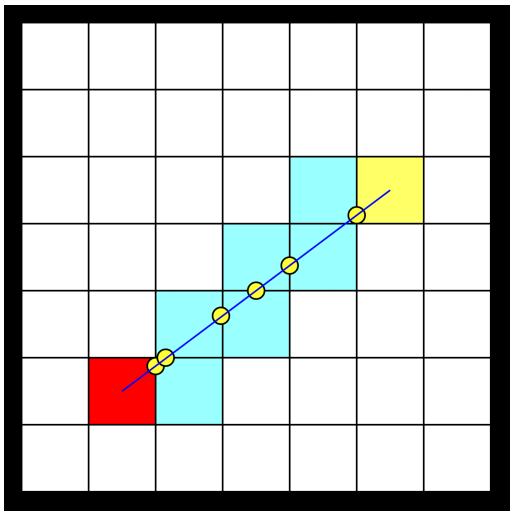
$$d(S,E) = \sqrt{(x_S - x_E)^2 + (y_S - y_E)^2} \tag{3.1}$$

Výpočet Euklidovské vzdálenost, kde parametry S a E značí počáteční, respektive koncový vrchol.

Algoritmus 3: Bresenhamův linkový algoritmus [27]

```
Input: x1, y1, x2, y2
dx ← abs(x2 - x1);
dy ← abs(y2 - y1);
sx ← 1 if x1 < x2 else -1;
sy ← 1 if y1 < y2 else -1;
err = dx - dy;
while x1 != x2 or y1 != y2 do
    if not maze.is_valid_position(x1, y1) or maze.is_wall(x1, y1) then
        return False
    e2 ← 2 * err;
    if e2 > -dy then
        err ← err - dy;
        x1 ← x1 + sx;
    if e2 < dx then
        err ← err + dx;
        y1 ← y1 + sy;
return True
```

Na Obrázku 3.3, lze vidět linku vedoucí od agenta (červený) do cílového vrcholu (žlutý) a vrcholy (modré), skrz které je třeba projít na určení zda v cestě linie se nenachází překážka a její potenciální kolizní body (žluté body).



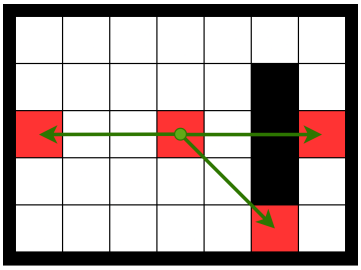
Obrázek 3.3 Linie vize a potenciální kolizní body

3.3.2 Komunikace

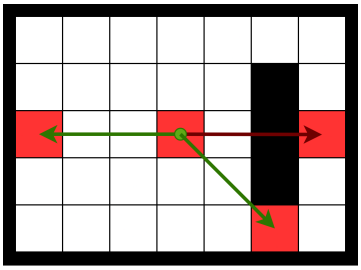
Pro komunikaci s ostatními agenty je potřeba, aby agent byl vybavený komunikačním modulem. Vybrané moduly se mezi sebou liší výpočtem vzdálenosti a maximálním dosahem, jinak nabízí společné rozhraní pro sdílení celé paměti nebo poslední aktualizace a snížit tak množství předaných dat. Mezi navržené způsoby řešení komunikace patří:

- Chybějící – Agent není vybavený komunikačním modulem
- Dosah přímou čarou – Vzdálenost mezi agenty se vypočítá jako Euklidovská vzdálenost [26]
- Přímá viditelnost (vizuální) – Pro určení zda agenti mohou komunikovat se použije upravený Bresenhamův Algoritmus 3
- Nejkratší cesta (zvukový) – Vzdálenost mezi agenty se rovná nejkratší cestě. Na snížení nutnosti opakovaných výpočtů A* je zde využito RRA* heuristiky (viz Kapitola 2.6.1)

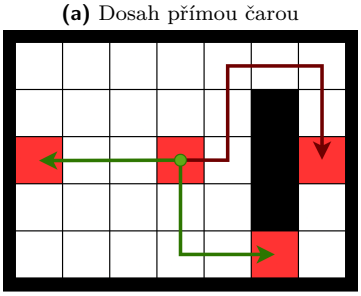
Na Obrázku 3.4 jsou vyobrazené vzdálenosti (šipky) mezi agenty (červené bloky). Zelené šipky určují úspěšnou komunikaci prostředního agenta s agentem, do kterého šipka vede, červené neúspěšnou komunikaci.



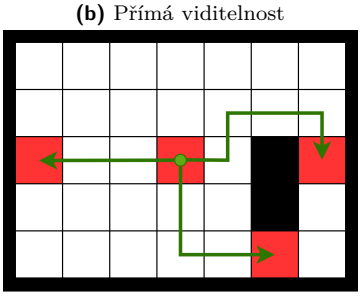
(a) Dosah přímou čarou



(b) Přímá viditelnost



(c) Nejkratší cesta s dosahem 5



(d) Nejkratší cesta s dosahem 5, menší překážka

Obrázek 3.4 Porovnání komunikačních způsobů

3.3.3 Rozhodovací algoritmus

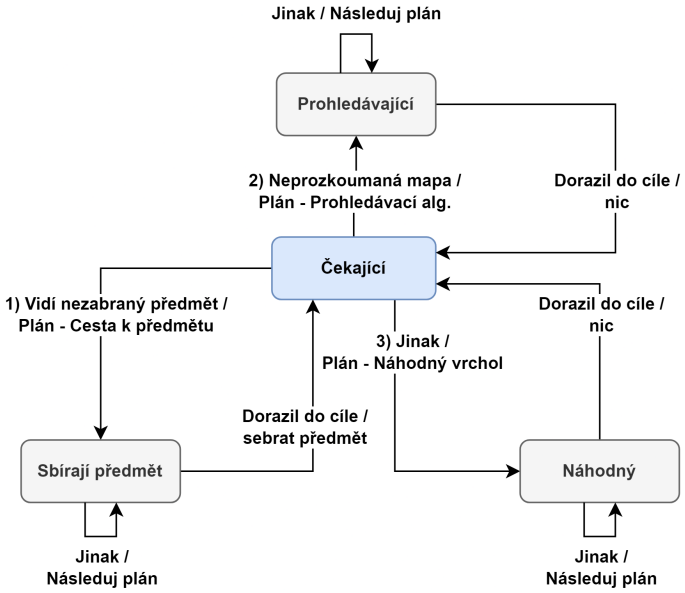
Jako způsob rozhodování agenta byly zvoleny *state-machines* neboli stavové stroje. Tento způsob rozhodování, byl již úspěšně implementovaný v různých druzích úkolů hledání cílů/objektů v neznámém prostředí. [28, 29, 30] A vzhledem k jednoduchosti implementace a jasnosti při rozhodování byl zvolen jako vhodný kandidát. Jednotlivé stroje jsou tvořeny manuálně.

Na Obrázku 3.5 je vidět základní rozhodovací stavový stroj. Bloky označují stav ve kterém se agent nachází. Modrý blok označuje výchozí stav, ve kterém se agent ocitne na počátku. V každém kroku *pre_step* agent rozhodne na základě jeho současném stavu a informací z vědomosti, jaký následující krok vykoná a jestli bude třeba vytvořit nový plán cesty.

Přechody mezi vrcholy se značí jako linky s počátkem a koncem v některém ze stavů stavu. Hodnoty [<**pořadí**>)] **podmínka / akce** označují v jakém pořadí a za jaké podmínky se dané přechody dají provést. Při splnění podmínky následně agent vykoná příslušnou akci. Automat v rámci kroku může přejít mezi více stavy, zastaví se až v době, kdy agent má plán nebo žádná z podmínek nebude splněna (Z pravidla, když byl zvolený přechod s podmínkou *Jinak*).

Za cenu výpočetního výkonu se dá přidat akce přeplánování cesty po každém kroku na základě nových informací. Díky tomu lze najít lepší cesty k danému vrcholu přes nově objevené úseky, které předtím byly neznámé a navigační algoritmus je tedy bral jako překážku. Dále případně podmínky, zda účel za vytvořeným plánem je stále potřebný/aktivní. Například, pokud agent jde směrem k předmětu a v době, kdy se dostane do dosahu senzorů a zjistí, že daný předmět je již sebraný, poté je výhodnější jen aktualizovat ve vědomosti stav předmětu a zvolit jiný cíl.

Použité stavové stroje, pro jednotlivé typy agentů, budou krátce popsané v rámci experimentální části.



Obrázek 3.5 Základní rozhodovací stavový stroj agenta

3.3.4 Prohledávací algoritmy

Mezi zvolené prohledávací algoritmy patří hladový prohledávací algoritmus, důkladněji popsaném v Kapitole 2.5.2. Pseudokód je vidět v Algoritmu 4. Proti klasickému `BFS` se liší tím, že na místo porovnání s cílem na Řádku 6, zjišťuje zda vrchol je neprozkoumaný.

Algoritmus 4: Greedy prohledávací algoritmus

```
   Input: start
1  QUEUE ← [start];
2  visited ← [];
3  prev ← ;
4  while QUEUE not empty do
5      node ← QUEUE.pop();
6      if node is unexplored then
7          return node;
8      neighbors ← node.valid_neighbors;
9      for neighbor ∈ neighbors do
10         if neighbor ∉ visited then
11             QUEUE.add(neighbor);
12             prev[neighbor] = node;
13             visited.add(neighbor);
```

Jako druhý algoritmus byl zvolen `NFA`, popsaný v Kapitole 2.5.3. Pseudokód části tvořící list hranic je popsaný v Algoritmu 5. Kde Q_e, Q_f představují frontu na kandidáty hraničních vrcholů a frontu na vrcholy, které mohou patřit do stejné hranice. Algoritmus začíná ve vstupním vrcholu a prochází sousedící, dokud nenarazí na hraniční.

Nad tímto hraničním vrcholem se následně spustí vnitřní část algoritmu, která poskládá komponentu hranice s daným vrcholem a vrátí ji (viz Algoritmus 6). Po vyčerpání všech dosažitelných vrcholů, Algoritmus 5 vrátí všechny existující hranice.

Algoritmus `NFA` následně vybere tu hranici, která je nejblíže počátečnímu vrcholu. Pro každou hranici se vypočítá těžiště za účelem vybrání nejvhodnější polohy, u které dojde k případnému pokrytí nejvíce hraničních vrcholů. Podle výrazu

$$(x, y) = \frac{1}{n}\left(\sum_{i=0}^{n} x_i, \sum_{i=0}^{n} y_i\right) \tag{3.2}$$

kde (x_i, y_i) je vrchol z hranice.

Nejbližší těžiště od počátku se následně určí pomocí Euklidovské vzdálenosti. [26] U zvolené hranice se ještě provede nalezení nejbližšího vrcholu, neboť není zaručené, že vrchol těžiště množiny bude průchozí/dosažitelný.

Jako třetí byl zvolen `WFD` algoritmus upravený, tak aby bral v potaz cíle ostatních agentů a volil následující hraniční vrchol tak, že na základě přednastavených pravděpodobnostech zvolí jiný vrchol v hranici nebo úplně jinou hranici, pokud je zvolený vrchol v okolí minimální vzdálenosti cíle jiného agenta.

Algoritmus 5: WFD [31]

```
   Input: start
1  Q_e ← [start];
2  visited ← {};
3  frontiers ← [];
4  while Q_e not empty do
5      node ← Q_e.pop();
6      if node ∈ visited then
7          continue;
8      visited.add(node);
9      if node is frontier then
10         frontiers.add(WFD_inner(node));
11     neighbors ← node.valid_neighbors();
12     for neighbor ∈ neighbors do
13         if neighbor ∉ visited then
14             Q_e.add(neighbor);
15 return frontiers
```

Algoritmus 6: WFD vnitřní část [31]

```
   Input: node
1  Q_f ← [node];
2  new_frontier ← [];
3  visited.remove(node);
4  while Q_f not empty do
5      cur ← Q_f.pop();
6      if cur ∈ visited then
7          continue
8      visited.add(cur);
9      new_frontier.add(cur);
10     neighbors ← cur.valid_neighbors();
11     for neighbor ∈ neighbors do
12         if neighbor ∉ visited and neighbor is frontier then
13             Q_f.add(neighbor);
14 return new_frontier;
```

3.4 Vstup

Program očekává na vstupu soubor scénáře. Scénář je v následujícím formátu. Na prvním řádku je volitelný **typ scénáře**, který určuje typ sbíraných předmětů a cíle spolupráce (Podrobnější popis v Kapitole 3.1). Možnosti jsou v rozmezí 1-4. Při neuvedení se bere typ jako typ 1.

1. Bludiště s předměty, kde se očekává, že každý agent může sebrat každý předmět
2. Bludiště, kde je část agentů omezena na činnost prohledávání
3. Bludiště, kde předměty mají omezení, který agenti je mohou sebrat, a u kterých je brán jako splněný/ukončený
4. Bludiště, kde jsou ještě přidány cílové vrcholy, kam agenti mají předměty odnést

Následuje konfigurace mapy. Na prvním řádku se nacházejí hodnoty **šířka X** a **výška Y** mapy. Samotná mapa je dále tvořena **Y** řádky o **X** sloupcích symbolů, kde symbol # značí stěnu/překážku a _ volný prostor.

Pokud následující neprázdná řádka po konfiguraci mapy začíná na *setups* a číslo **C**, tak poté následuje **C** nastavení rolí agentů. Každé nastavení je tvořené řádkou s počtem upravovaných hodnot a číslo role, o které se jedná. Možnosti typů a jejich popis je znázorněn v Tabulce 3.1. Možné hodnoty jsou popsané dále v příloze programu.

Tabulka 3.1 Možnosti konfigurace role

zkratka	typ	dodatečná hodnota	příklad
com	komunikace	číslo – vzdálenost	c 5
sen	senzor	číslo – vzdálenost	r 5
sym	symbol role	znak – symbol	T
exp	prohledávací algoritmus	—	nfa
dec	Rozhodovací stroj chování	—	s3_helper

Další řádek určuje počet agentů. Každý agent je reprezentovaný mezerou oddělenými hodnotami (**X, Y**) – počáteční pozice a nepovinnou hodnotou **R** – role agenta, která je při vynechání rovna 0 (agent má roli 0).

Po určení agentů je na dalším řádku počet předmětů. Formát vstupu předmětů je závislý na typu scénáře. Předměty lze vygenerovat na vybraných místech nebo náhodně. Formát pro vybrané místa je následující:

- typ 1, 2, 4 – trojice – (**X Y P**) – (**X, Y**) pozice předmětu v mapě, **P** maska rolí, které předmět mohou sebrat
- typ 3 – čtveřice – (**X Y P T**) – **T** navíc značí masku rolí, které jsou cílové/ukončující

A pro náhodné:

- typ 1, 2, 4 – dvojice – (**C P**) – **C** počet předmětů, **P** maska rolí, které předmět mohou sebrat
- typ 3 – trojice – (**C P T**) – **T** navíc značí masku rolí, které jsou cílové/ukončující

Za hodnoty pro náhodné rozložení je možné ještě přidat čtveřici čísel (**X1, Y1, X2, Y2**) – (**X1, Y1**) pozice levého horní rohu a (**X2, Y2**) pozice pravého dolního rohu, kde se generované předměty omezí na prostor mezi daným rozmezím.

Pokud zvolený typ scénáře je 4, tak ještě na konci je potřeba řádek s počtem cílů a následující řádky s hodnotami (**X, Y**) jejich pozicemi.

Na Obrázku 3.6 je ukázka možného zápisu scénáře. Soubor lze doplnit o komentáře pomocí `//`, přidat prázdné řádky nebo tabulátory pro lepší čitelnost. Následně na Obrázku 3.7 je výsledné náhodné počáteční rozložení scénáře. Kde symbol `@` značí pozici obsazenou předmětem, `#` stěnou, `A` agentem, `T` cílem a symbol `_` volný prostor.

```
type 4              // scénář 4
10 5                // rozměr mapy (10, 5)

----------
####__####
----------
----------
----------

setups 1
    0 4             // nastavení role 0
    sen r 5         // senzor kruhový 5
    com c 5         // komunikace dosah  5
    exp nfa         // prohledávací algoritmus NFA
    sym A           // symbol role "A"

1       // agenti
0 0 0           // agent bez role na pozici (0,0)

1       // 5 náhodných předmětů mezi (0, 2) a (10, 5)
r 5 0 0 2 10 5

2       // cíle
4 1
5 1
```

Obrázek 3.6 Příklad textové reprezentace scénáře

```
A_________
####TT####

----------
___@@__@__
@____@____
```

Obrázek 3.7 Příklad výsledného rozložení mapy z Obrázku 3.6

3.5 Výstup

Výstupem programu je textová vizualizace stavu mapy v každém kroku simulace. Přes `scenario_file` se zvolí simulovaný soubor scénáře a přes `output_file` výstupní soubor pro experimentální výsledky.

V rámci vizualizace lze nastavit ID sledovaného agenta `followed_agentID` (agent, jehož stav paměti/senzorů bude zobrazován). Pokud není nastavený, tak program zobrazí kolektivní sjednocenou paměť/senzory všech agentů. Lze ještě zvolit mlhu `fog`, která v případě `False` způsobí, že program bude zobrazovat celou mapu jako odhalenou s známými pozicemi agentů, předmětů a cílů.

Jako vedlejší výstup pro experimentování je zápis průběhu dosažení cílů simulace. Pomocí příznaku `exploration_only` se dá nastavit, že cílem simulace je pouze prohledat celou mapu, jinak je cílem je úspěšné sebrání/splnění všech předmětů.

Kapitola 4

Experimentální výsledky

V této kapitole jsou popsány výsledky experimentů. Experimenty probíhají tak, že na vytvořených mapách se opakovaně spustí simulace s nastavením agentů ve výchozím nastavení. Výjimkou jsou rozdíly v testovaných částech.

4.1 Výchozí nastavení agenta

Výchozí nastavení agenta lze najít v Tabulce 4.1

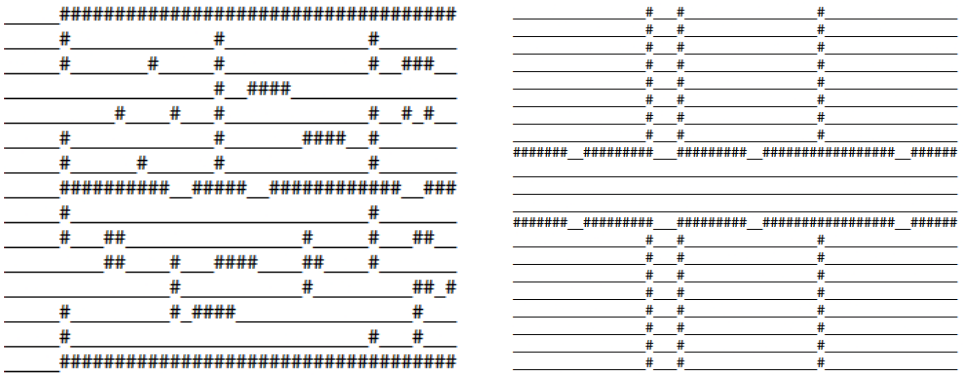
Tabulka 4.1 Výchozí nastavení agenta

Parametr	Hodnota	Poznámka
Senzor	Kruhový 5	Kruhový senzor s dosahem 5 vrcholů
Komunikace	Dosah 5	Dosah přímou čarou s dosahem 5 vrcholů
Role	0	Agent může interagovat pouze s předměty bez omezení
Prohledávací alg.	Greedy	V každém kroku prohledávání volí ten nejbližší vrchol
Rozhodování	Úvodní	Prohledává prostředí, dokud nenajde předmět, který se následně pokusí hned sebrat, jinak směruje na náhodný známý vrchol

4.2 Použité mapy na experimentování

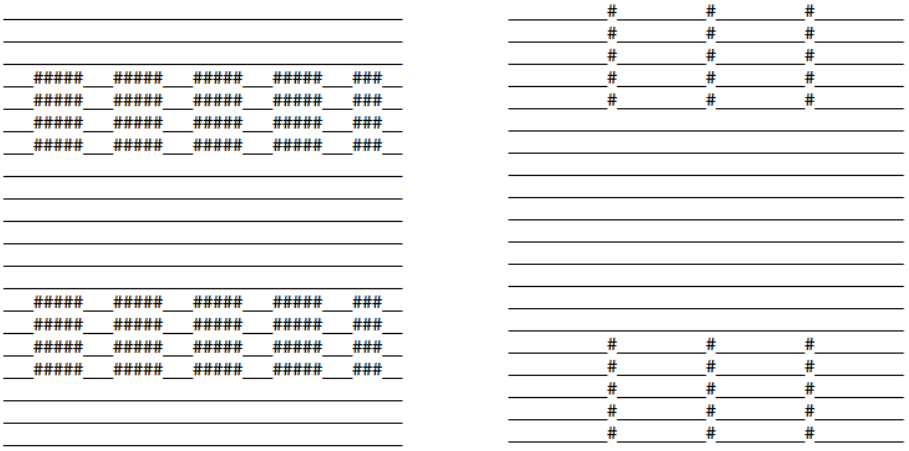
Pro experimentování byly vytvořeny mapy, které se liší v jejich otevřenosti a množství překážek. Dále byly použity mapy převzaté z [20] a jejich varianty. Varianty představují převážně užší verze map, ve kterých jsou redukované otevřené prostory.

Mapa domu na Obrázku 4.1a představuje menší prostor s velkým množstvím překážek a obsahuje málo otevřeného prostoru. Parkoviště a budova na Obrázku 4.2b, respektive 4.1b představují otevřenější prostory, kde budova se dále dělí na jednotlivé otevřené místnosti. Mapa skladiště na Obrázku 4.2a kombinuje otevřený prostor ve středu prostředí a mnoho uliček mezi překážkami.



(a) Dům **(b)** Budova

Obrázek 4.1 Vlastní mapy



(a) Skladiště **(b)** Parkoviště

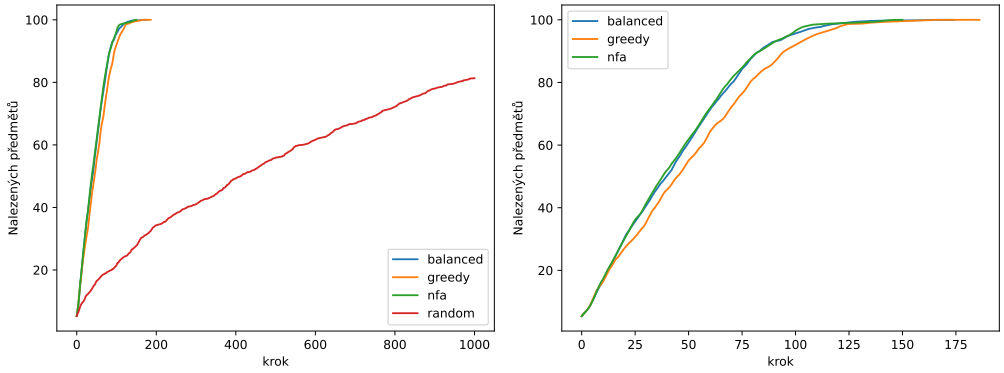
Obrázek 4.2 Mapy převzaté z [20]

4.3 Experimenty s rychlosti prohledání

Rychlost prohledávání byla testována na skupině agentů o počtu 3 ve výchozím nastavení, s rozdílnou volbou prohledávacího algoritmu.

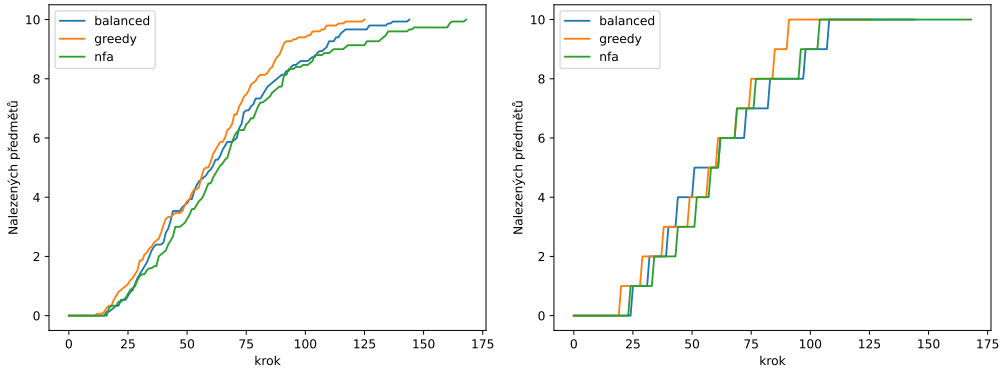
Z grafu na Obrázku 4.3a lze vyčíst, že náhodná procházka není ideální pro rychlé prohledání bludiště, ale i přes to zvládne v krokovém limitu prozkoumat >50 % prostředí. Z grafu na Obrázku 4.3b je viditelné, že zvolené algoritmy jsou ve svých rychlostech prohledání podobné. Pro případné výraznější zlepšení balancovaného *Frontier* algoritmu, by bylo potřeba lepší balancování agentů při volbě jejich cíle (třeba použitím popsaného `BSO` v Kapitole 2.5.3.3).

Pro experiment k ověření rychlosti byla zvolena mapa domu a na ní v 15 iteracích spuštěna simulace s úkolem sebrání náhodně umístěných předmětů v rámci mapy, s cílem dosáhnout statisticky spolehlivého výsledku. Výsledky lze vidět na grafech na Obrázku 4.4. Výsledky jsou podobné výsledkům při porovnávání rychlosti algoritmů, jen je zde vidět, že při použití *greedy* algoritmu je splnění úkolu výrazně rychlejší.



(a) Včetně procházky - průměr **(b)** Bez procházky - průměr

Obrázek 4.3 Porovnání rychlosti prohledávání přes 3 různé mapy



(a) Průměr **(b)** Medián

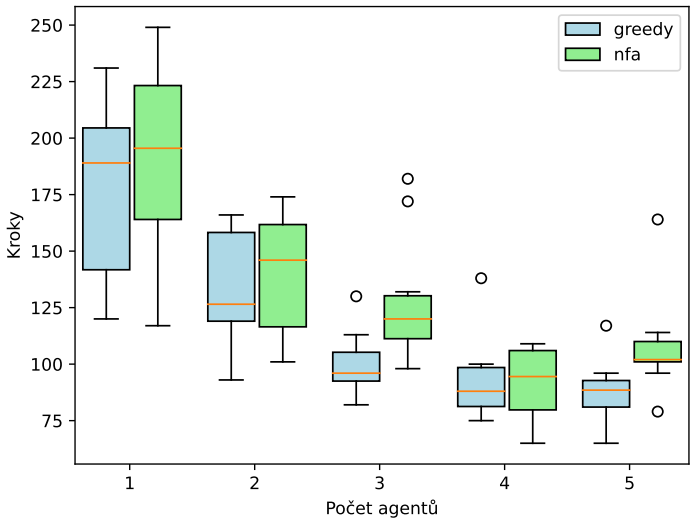
Obrázek 4.4 Porovnání rychlosti sebrání 10 předmětů na mapě domu

4.4 Experimenty s měnícím se počtem a typy agentů

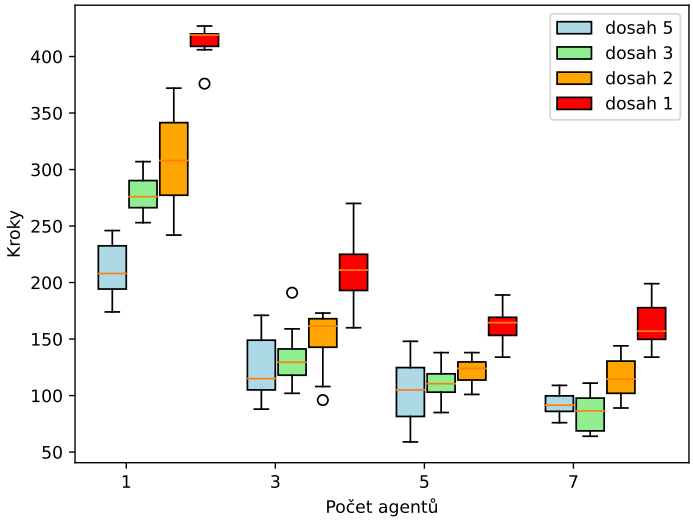
Další experimenty jsou zaměřeny na sledování vlivu počtu agentů. Na grafu z Obrázku 4.5 lze vidět, jak s přibývajícím počtem agentů klesá střední hodnota a rozptyl počtu kroků na sebrání všech předmětů nezávisle na zvoleném algoritmu prohledávání.

S tvarem mapy (uzavřenější menší místnosti) se simulace chová dle očekávání. S přibývajícím počtem agentů je přidaná hodnota čím dál menší. Proti `NFA` má `greedy` výhodu rychlého rozdělení agentů na začátku, kde proti tomu u `NFA` je větší šance zvolení stejného vrcholu na prohledání. Tím algoritmus dosáhl většího zkrácení počtu kroků již u přidání druhého agenta.

Na Obrázku 4.6 lze proti tomu vidět závislost počtu kroků potřebných na nalezení předmětů vůči počtu agentů a jejich dosahu vizuálních senzorů. Při simulaci s nastavením senzoru s dosahem 1 se jedná o klasické prohledání jen sousedících vrcholů.



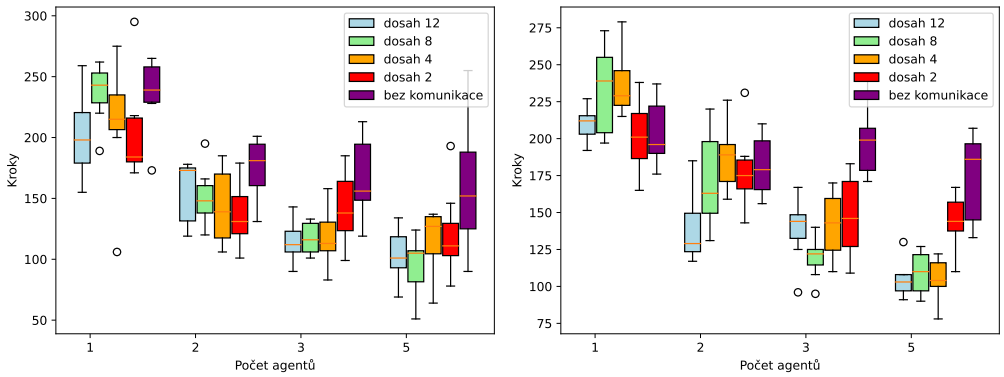
Obrázek 4.5 Porovnání rychlosti sebrání 10 předmětů na mapě domu v závislosti na počtu agentů



Obrázek 4.6 Porovnání rychlosti sebrání 10 předmětů na mapě domu v závislosti na volbě dosahu senzorů. Dosah komunikace navíc omezený na 5.

Na Obrázku 4.7 jsou grafy ukazující vliv počtu agentů a dosahu komunikace na rychlosti prohledání mapy. U příkladu na Obrázku 4.7a je vidět, že volba dosahu komunikace (kromě žádné) nemá statisticky výrazný vliv na počet kroků. U skladiště, které má více uliček a velký otevřený prostor uprostřed je vidět jistá preference pro větší dosahy, které mají menší rozptyl a střední hodnotu proti nízkému dosahu komunikačního modulu.

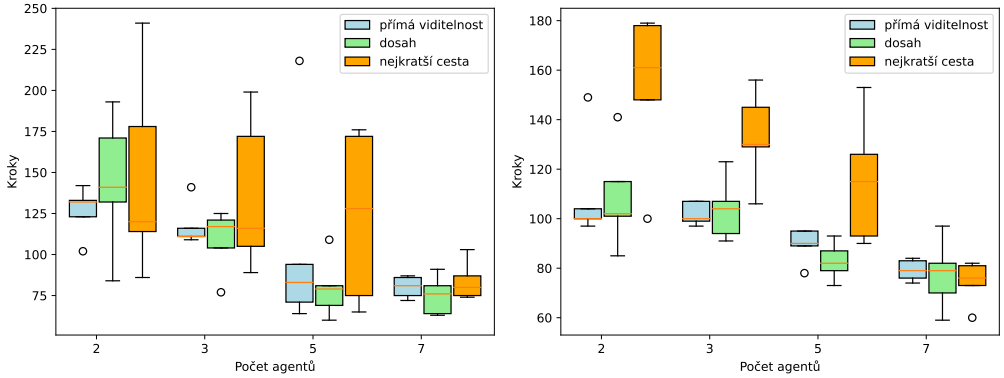
Na grafech na Obrázku 4.8, jde vidět jaký vliv má zvolený typ komunikace na rychlost sebrání předmětů. Proti přímé vzdálenosti a dosahu je vidět, že nejkratší cesta ztrácí. Faktem ale je, že při použití A* jen se 4 směry pohybu bude mít při stejném dosahu senzoru, bude mít tento způsob komunikace nižší efektivní vzdálenost na diagonálách.



(a) Dům **(b)** Skladiště

Obrázek 4.7 Porovnání rychlosti sebrání 10 předmětů na mapách domu a skladiště v závislosti na volbě dosahu komunikačního modulu

Na mapě domu, kde je prostor více uzavřený, je vidět, že rozdíl mezi dosahem a přímou viditelností není výrazný. Toto platí i u mapy parkoviště, která je velký otevřený prostor s malým počtem překážek, kde podle očekávání, viditelnost nemá velký vliv na potřebný počet kroků.



(a) Dům **(b)** Parkoviště

Obrázek 4.8 Porovnání rychlosti sebrání předmětů v závislosti na volbě typu komunikačního modulu

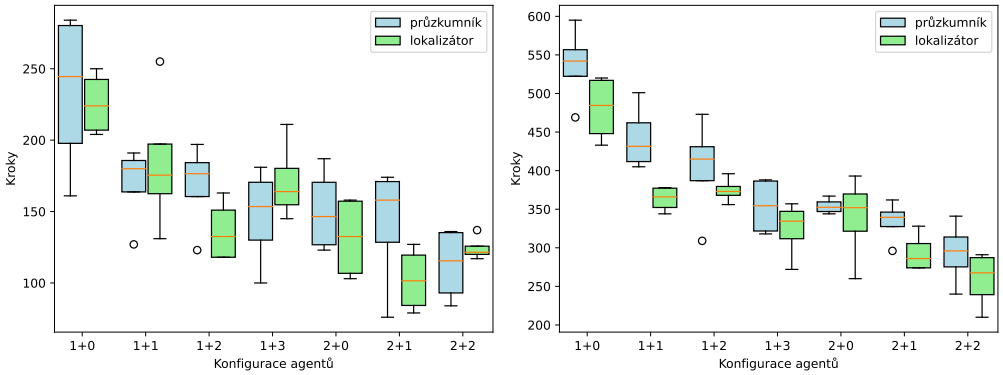
4.5 Experimenty s techniky a pomocníky

U scénářů, kde se agenti dělí do dvou skupin. `Technici` – dokážé interagovat s předměty a `pomocníci` – pouze sdílejí informace ohledně průzkumu. Bylo využito úvodní chování pro techniky a pro pomocníky se udělali dvě různé. Jedno, které se snaží prohledat celé prostředí a pak následně chodit náhodně. Dále se bude nazývat `průzkumník`.

Druhý typ po prohledání vyrazí na místa, kde naposledy viděl agenta, který by byl schopný interagovat s předmětem, na kterém ještě nebyla provedená akce. Tento agent bude pod názvem `lokalizátor`. Pro případ, kdy agent vyrazí na posledního

cíle sledovaného agenta jsou výsledky podobné. Experimenty byly provedeny na mapě domu a budovy. Dům představuje menší prostředí, ale s více překážkami a budova větší a otevřenější prostředí. Na Obrázku 4.9 jsou vidět grafy s porovnáním kombinací počtů agentů jednotlivých typů.

Na Obrázku 4.9b je vidět, že na větších mapách náhodná volba začíná ztrácet proti sofistikovanějším volbám cílu.



(a) Dům **(b)** Budova

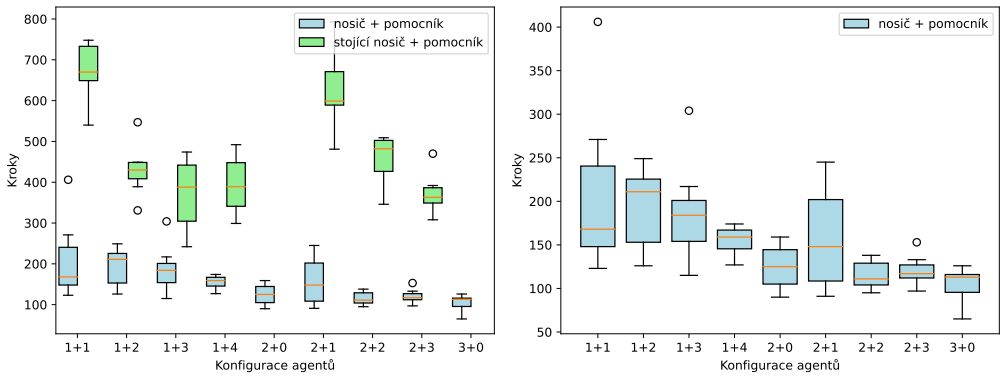
Obrázek 4.9 Porovnání rychlosti sebrání předmětů na mapách domu a budovy při kooperaci pomocníků a techniků

4.6 Experimenty s nosiči a pomocníky

Ve scénářích pro `nosiče` a `pomocníky` se experimentovalo s tím, že buď nosiči budou stát na místě nebo se budou pohybovat a pomáhat tak s prohledáváním či sběrem předmětů. Při stání se tento problém redukuje na scénář typu 4. Při pohybu bylo potřeba pomocníkům přidat možnost navigovat se k pohyblivým cílům. Podobně jako `lokalizátor` se bude směrovat na poslední známé pozice daného agenta.

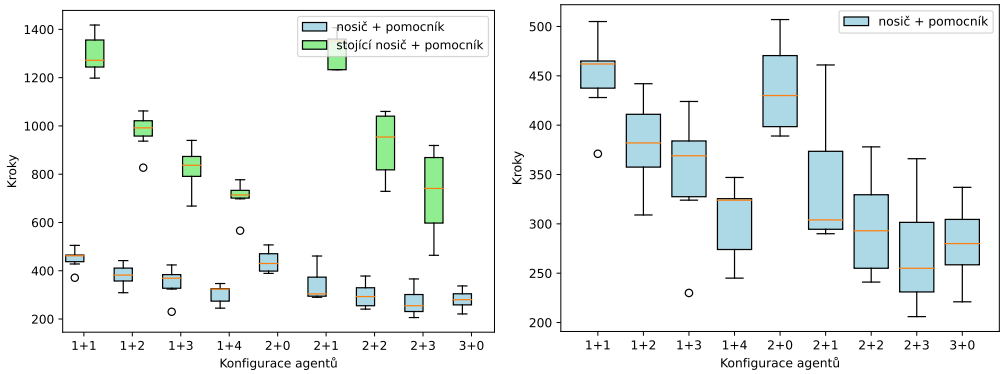
Na Obrázku 4.10 a 4.11, lze vidět jak rychle jednotlivé kombinace počtů nosičů a pomocníku dokázali sesbírat jednotlivé předměty. Z výsledku jde vidět, že při jednom pomocníku jsou výsledky nejvíce roztažené. To je způsobené převážně tím, když pomocník najde předmět a následně se ho snaží odnést nosiči, ale má ho problém najít a ztrácí tak drahocenné kroky.

S přibývajícím počtem agentů klesá potřebný čas a přidání jednoho nosiče má podobný efekt jako přidání dvou pomocníků.



(a) Dům **(b)** Dům bez stojících

Obrázek 4.10 Porovnání rychlosti sebrání předmětů na mapě domu při kooperaci nosičů a pomocníků



(a) Budova **(b)** Budova bez stojících

Obrázek 4.11 Porovnání rychlosti sebrání předmětů na mapě budovy při kooperaci nosičů a pomocníků

4.7 Experimenty odnošení předmětů

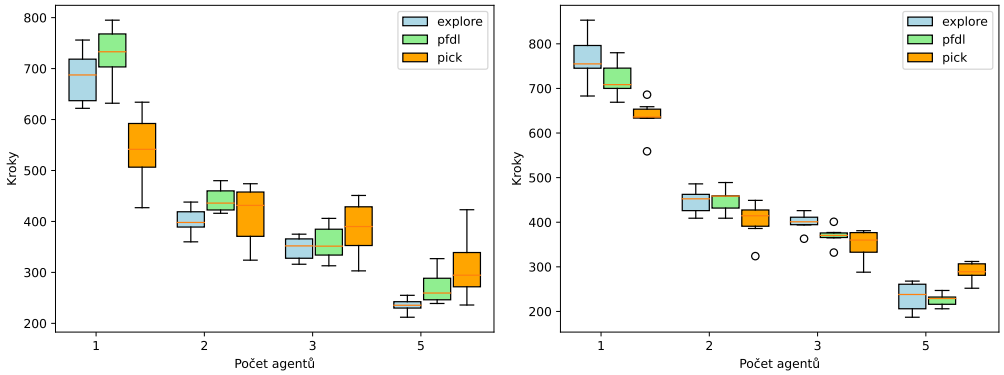
Pro scénáře na odnošení předmětů, byly vytvořeny tři stavové stroje, které se liší v reakci na nalezení předmětu.

- *pick-first* – Agent při nalezení předmětu ihned předmět zvedne a odnese na cílové místo a následně se vrátí k prohledávání
- *explore-first* – Agent nezvedá předměty dokud nemá prohledanou celou mapu.
- *pick-first-deliver-later* – `PFDL` – neboli seber co nejdříve a dones později, kde agent zvedne první nalezený předmět, ale odnese ho na místo až tehdy, kdy bude prozkoumané celé prostředí.

PFDL je takovým mixem mezi pick-first a explore-first a v jistých případech by mohl mít hypoteticky lepší výsledky než oba zmíněné. Ale už v rámci návrhu byl vidět jasný (a častý) případ, ve kterém bude mít výrazně horší výsledek. Stačí aby jeden předmět

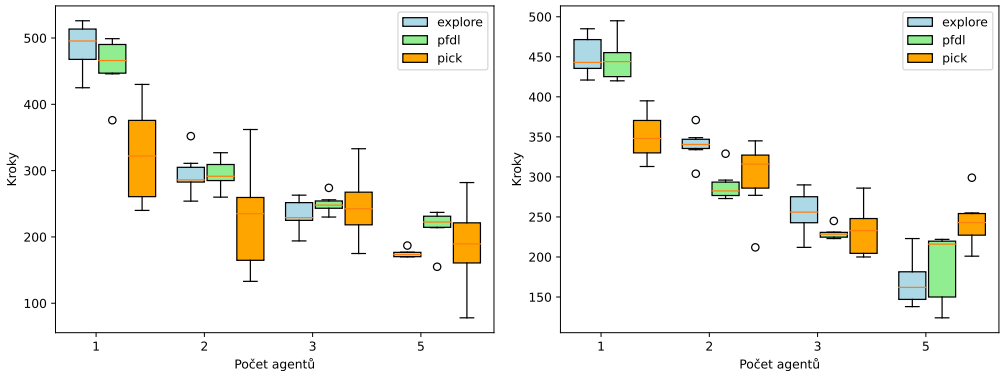
byl na začátku prohledávání a k jakémukoli předmětu na konci bludiště bude třeba jít 2x.

Na Obrázku 4.12 jsou vidět vlivy zvoleného stavového stroje rozhodování při 10 předmětech na mapě a na Obrázku 4.13 v případě pouze 5 předmětů na mapě. Je vidět, jak v nízkém počtu agentů má pick-first rychlostní převahu proti ostatním, ale tato výhoda vymizí s přibývajícím počtem, až se do převahy dostane explore-first.



(a) Dům **(b)** Skladiště

Obrázek 4.12 Porovnání způsobů sesbírání a odnesení 10 předmětů

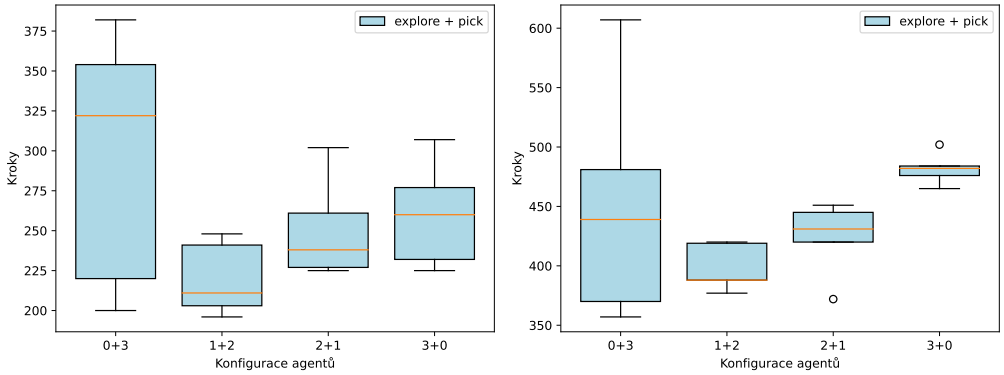


(a) Dům **(b)** Skladiště

Obrázek 4.13 Porovnání způsobů sesbírání a odnesení 5 předmětů

Dále se experimentovalo jestli se nedá vylepšit rychlost zpracování předmětů, použitím různých kombinací agentů. Testovali se různé kombinace pro skupinu o velikosti 3 a 4. U velikosti 5+ už dochází k saturaci na zvolených mapách a výsledky se od sebe moc neliší nezávisle na zvoleného chování.

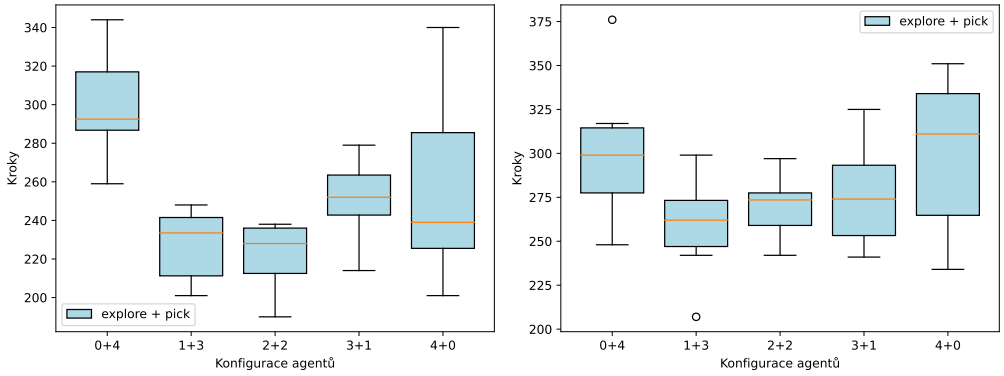
Na Obrázku 4.14 jsou vidět grafy počtu kroků pro kombinace u 3 agentů. Je vidět, že s přibývajícím počtem nejdříve prohledávajících agentů, roste průměr potřebných kroků na odnošení předmětů. U všech agentů jako nejdříve odnášející je, ale proti tomu vidět velký rozptyl v závislosti na rozložení předmětů v prostředí.



(a) Dům **(b)** Skladiště

Obrázek 4.14 Porovnání kombinací 3 agentů při přenášení předmětů

Na Obrázku 4.15 jsou grafy, které ukazují, že při 4 agentech je na mapě domu 4.15a nejrychlejší mix 2:2 a extrémy, kde jsou agenti jednoho typu, jsou méně vhodné a mají větší rozptyl. Proti tomu na mapě skladiště 4.15b jsou si kombinace agentů podobné a rozdíl mezi 2:2 a extrémy není tak výrazný.



(a) Dům **(b)** Skladiště

Obrázek 4.15 Porovnání kombinací 4 agentů při přenášení předmětů

Kapitola 5

Závěr

Byla provedena literární rešerše na téma algoritmů na prozkoumávání a přiřazování cílů agentům. Některé algoritmy byly zvoleny pro následné použití při návrhu implementace simulačního prostředí na řešení problému kolaborace agentů při sběru předmětů v neznámém bludišti.

Posléze byla navržena série různých příkladů, které se lišili ve způsobu kooperace agentů. Tyto příklady byly opatřeny variantami s rozdílnými vlastnostmi agentů. Vytvořené scénáře byly následně otestovány na vytvořeném modelu a porovnali se rychlosti splnění cílů při použití různých kombinací chování agentů a vlastností agentů.

Model se dá dále použít na zkoušení chování agentů a případné rozhodování o použití skupin agentů s cílem optimalizovat čas zpracování všech předmětů, dle podmínek scénáře. Model pracuje s předpokladem perfektního snímání a bezproblémové komunikace, možným podnětem na navázání je tedy přidání nejistoty do snímání senzorů nebo použití více sofistikovaných způsobů pro komunikaci a přiblížit se tak více reálnému použití.

Do modelu se dají přidat další způsoby chování, ale zvolený způsob už při větších rozhodovacích stavových strojích začal ukazovat své problémy s udržovatelnosti a čitelností, proto dalším námětem na vylepšení, by mohl být přechod na stromy chování nebo využití jiného vzoru na strukturování a dovolit tak větší znovupoužitelnost kusů operací agenta.

Bibliografie

1. PARASUMANNA GOKULAN, Balaji; SRINIVASAN, D. An Introduction to Multi-Agent Systems. In: 2010, sv. 310, s. 1–27. ISBN 978-3-642-14434-9. Dostupné z DOI: 10.1007/978-3-642-14435-6_1.
2. VLASSIS, Nikos. A concise introduction to multiagent systems and distributed artificial intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning*. 2007, roč. 1, č. 1, s. 1–71.
3. RUSSELL, Stuart J. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.
4. BANERJEE, Bikramjit. Autonomous Acquisition of Behavior Trees for Robot Control. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, s. 3460–3467. Dostupné z DOI: 10.1109/IROS.2018.8594083.
5. CHANG, Kyong-Sok; ZHU, David. *2.4 Hierarchical Finite State Machine (HFSM) & Behavior Tree (BT)* [online]. 2020. [cit. 2023-05-08]. Dostupné z: https://web.stanford.edu/class/cs123/lectures/CS123_lec08_HFSM_BT.pdf.
6. Reinforcement learning: A survey. *Journal of artificial intelligence research*. 1996, roč. 4, s. 237–285.
7. BAKER, Bowen; KANITSCHEIDER, Ingmar; MARKOV, Todor M.; WU, Yi; POWELL, Glenn; MCGREW, Bob; MORDATCH, Igor. Emergent Tool Use From Multi-Agent Autocurricula. *CoRR*. 2019, roč. abs/1909.07528. Dostupné z arXiv: 1909.07528.
8. ELFES, A. Using occupancy grids for mobile robot perception and navigation. *Computer*. 1989, roč. 22, č. 6, s. 46–57. Dostupné z DOI: 10.1109/2.30720.
9. COLLINS, Thomas; COLLINS, J.J.; RYAN, Donor. Occupancy grid mapping: An empirical evaluation. In: *2007 Mediterranean Conference on Control & Automation*. 2007, s. 1–6. Dostupné z DOI: 10.1109/MED.2007.4433772.
10. LI, Gao; ZHANG, Dabu; SHI, Yuhui. An Unknown Environment Exploration Strategy for Swarm Robotics Based on Brain Storm Optimization Algorithm. In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. 2019, s. 1044–1051. Dostupné z DOI: 10.1109/CEC.2019.8789994.

11. IDRIES, Mohamed Osama; ROLF, Matthias; SCHEPER, Tjeerd V olde. Exploration: Do We Need a Map? In: *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019, London, UK, July 3–5, 2019, Proceedings, Part II 20*. Springer, 2019, s. 476–479.
12. NTAWUMENYIKIZABA, Abdallah; VIET, Hoang Huu; CHUNG, TaeChoong. An online complete coverage algorithm for cleaning robots based on boustrophedon motions and A* search. In: *2012 8th International Conference on Information Science and Digital Content Technology (ICIDT2012)*. 2012, sv. 2, s. 401–405.
13. BLACK, Paul E. Greedy algorithm. *Dictionary of Algorithms and Data Structures* [online]. 2005, roč. 2, s. 62 [cit. 2021-05-03]. Dostupné z: `https://www.nist.gov/dads/HTML/greedyalgo.html`.
14. TOVEY, Craig; KOENIG, Sven. Improved analysis of greedy mapping. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. IEEE, 2003, sv. 4, s. 3251–3257.
15. KOENIG, Sven; TOVEY, Craig; HALLIBURTON, William. Greedy mapping of terrain. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*. IEEE, 2001, sv. 4, s. 3594–3599.
16. WAGNER, I.A.; LINDENBAUM, M.; BRUCKSTEIN, A.M. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*. 1999, roč. 15, č. 5, s. 918–933. Dostupné z DOI: `10.1109/70.795795`.
17. YAMAUCHI, Brian. A frontier-based approach for autonomous exploration. In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 1997, s. 146–151.
18. HOLZ, Dirk; BASILICO, Nicola; AMIGONI, Francesco; BEHNKE, Sven. Evaluating the Efficiency of Frontier-based Exploration Strategies. In: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. 2010, s. 1–8.
19. BURGARD, Wolfram; MOORS, Mark; FOX, Dieter; SIMMONS, Reid; THRUN, Sebastian. Collaborative multi-robot exploration. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. IEEE, 2000, sv. 1, s. 476–481.
20. LI, Gao; ZHANG, Dabu; SHI, Yuhui. An Unknown Environment Exploration Strategy for Swarm Robotics Based on Brain Storm Optimization Algorithm. In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. 2019, s. 1044–1051. Dostupné z DOI: `10.1109/CEC.2019.8789994`.
21. SHI, Yuhui. Brain storm optimization algorithm. In: *Advances in Swarm Intelligence: Second International Conference, ICSI 2011, Chongqing, China, June 12-15, 2011, Proceedings, Part I 2*. Springer, 2011, s. 303–309.
22. HART, Peter E.; NILSSON, Nils J.; RAPHAEL, Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107. Dostupné z DOI: `10.1109/TSSC.1968.300136`.

23. BLACK, Paul E. Manhattan distance. *Dictionary of algorithms and data structures* [online]. 2019 [cit. 2023-05-03]. Dostupné z: `https://www.nist.gov/dads/HTML/manhattanDistance.html`.

24. SILVER, David. Cooperative pathfinding. In: *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. 2005, sv. 1, s. 117–122. Č. 1.

25. KOOPMAN, P. Bresenham line-drawing algorithm. *Forth Dimensions*. 1987, roč. 8, č. 6, s. 12–16.

26. BLACK, Paul E. Euclidean distance. *Dictionary of Algorithms and Data Structures. US National Institute of Standards and Technology* [online]. 2004 [cit. 2023-05-03]. Dostupné z: `https://www.nist.gov/dads/HTML/euclidndstnc.html`.

27. ALOIS, Zingl. *A rasterizing algorithm for drawing curves* [online]. 2012. [cit. 2023-05-03]. Dostupné z: `https://www.scribd.com/document/337616662/A-Rasterizing-Algorithm-for-Drawing-Curves-Alois-Zingl-2012-pdf`.

28. SIJPKENS, Thomas; SIMON, Eliott; LOGTENS, Tijn; SPARIDANS, Luc; THIL, Lucas-Andrei; MERSCH, Tobias. Multi-agent Surveillance Simulated by Means of Pursuit Evasion Games. 2020.

29. ALLISON, Mark; SPRADLING, Matthew; KNOCH, Nicholas. UAV Collaborative Search Using Probabilistic Finite State Machines. *ICCRTS-KSCO 2017*. 2017.

30. BÄHNEMANN, Rik; SCHINDLER, Dominik; KAMEL, Mina; SIEGWART, Roland; NIETO, Juan. A decentralized multi-agent unmanned aerial system to search, pick up, and relocate objects. In: *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, 2017, s. 123–128.

31. TOPIWALA, Anirudh; INANI, Pranav; KATHPAL, Abhishek. Frontier Based Exploration for Autonomous Robot. *CoRR*. 2018, roč. abs/1806.03581. Dostupné z arXiv: `1806.03581`.

Obsah přílohy